RL-TR-95-164, Vol I (of five)
Final Technical Report
September 1995

HIGH-LEVEL ADAPTIVE SIGNAL PROCESSING ARCHITECTURE WITH APPLICATIONS TO RADAR NON-GAUSSIAN CLUTTER

University of Massachusetts at Amherst

Victor R. Lesser (University of Massachusetts)
Hamid Nawab (Boston University)
Donald Weiner (Syracuse University)

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

RL-TR-95-164, Vol I (of five), has been reviewed and is approved for publication.

APPROVED: Vincent C. Vannicola

DR. VINCENT C. VANNICOLA
Project Engineer

FOR THE COMMANDER: [illegible]

DONALD W. HANSON
Director of Surveillance & Photonics

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE September 1995	3. REPORT TYPE AND DATES COVERED Final Apr 91 - Jun 94

4. TITLE AND SUBTITLE HIGH-LEVEL ADAPTIVE SIGNAL PROCESSING ARCHITECTURE WITH APPLICATIONS TO RADAR NON-GAUSSIAN CLUTTER	5. FUNDING NUMBERS C - F30602-91-C-0038 PE - 62702F PR - 4506 TA - 11 WU - 1B
6. AUTHOR(S) Victor R. Lesser (University of Massachusetts), Hamid Nawab (Boston University), and Donald Weiner (Syracuse University)	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Massachusetts at Amherst Department of Computer Science Lederle Graduate Research Center Amherst MA 01003	8. PERFORMING ORGANIZATION REPORT NUMBER N/A
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Rome Laboratory (OCSS) 26 Electronic Pky Griffiss AFB NY 13441-4514	10. SPONSORING/MONITORING AGENCY REPORT NUMBER RL-TR-95-164, Vol I (of five)

11. SUPPLEMENTARY NOTES

Rome Laboratory Project Engineer: Dr. Vincent C. Vannicola/OCSS/(315) 330-2861

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.	12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

We describe the accomplishments of a collaborative effort carried out by a team of researchers from the University of Massachusetts at Amherst (headed by V. Lesser), Boston University (headed by H. Nawab), and Syracuse University (headed by D. Weiner) on the development of a high-level signal processing architecture called IPUS (Integrated Processing and Understanding of Signals). Based on the IPUS generic testbed architecture and "radar and sound understanding" (RESUN) architectures, we have been able to transfer IPUS technology from a LISP environment to a C++ environment for use in an IPUS Radar Clutter Analysis Testbed. Though not as well-developed as the sound understanding application because of its newness, this radar testbed has still clearly demonstrated the potential of IPUS-like technologies for CFAR processing of radar returns. There has also been significant development of knowledge for weak signal detection. This knowledge has involved the application of the Ozturk algorithm to hypothesize the distribution of data in a clutter patch based on a small amount of data. Also, techniques have been developed for partitioning the radar surveillance volume into background noise and clutter patches, for weak signal detection in K-distributed clutter, and the efficient use of Rejection Theorem for Weibull clutter generation. Though the effort on the application of IPUS to communications was given less priority, we still did some interesting theoretical work on (see reverse)

14. SUBJECT TERMS Radar, Signal processing, Artificial intelligence detection. Clutter, Non-Gaussian	15. NUMBER OF PAGES 220
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

13. (Cont'd)

weak signal detection in communication systems subject to spherically invariant random processes (SIRP) interference.

Contents

List of Figures

List of Tables

Chapter 1

Executive Summary

In this report, we describe the accomplishments of a collaborative effort carried out by a team of researchers from the University of Massachusetts at Amherst (headed by V. Lesser), Boston University (headed by H. Nawab) and Syracuse University (headed by D. Weiner) on the development of a high-level adaptive signal processing architecture, called IPUS (*Integrated Processing and Understanding of Signals*).

The IPUS architecture is a framework for next-generation signal-interpretation systems. Current systems have most often assumed that fixed signal processing in the front-end can provide adequate (not necessarily optimal) evidence for reliable interpretations regardless of the range of possible scenarios in the environment. In our opinion, this assumption is plausible for systems that monitor stable environments, but not for those that monitor complex environments. In these environments, the choice of front-end signal processing algorithms (SPAs) is crucial to the generation of adequate evidence for interpretation processes. Front-end SPA sets for complex environments must be dynamically modifiable to respond to scenario changes and to reprocess ambiguous or distorted data. The term "dynamically modifiable" refers both to the ability to change a particular SPA's control parameter values and to the ability to select entirely new sets of front-end SPAs.

The long-term goals for the project set out in the contract were:

- Refine IPUS architecture and control structure
- Extend architecture to handle multiple sensor types
- Investigate architecture's effectiveness:
  - increased knowledge base size
  - decreased signal/noise ratio
  - increased scenario complexity
- Apply the signal re-processing architecture in statistical signal processing domains (e.g. radar and communications)

- Determine architecture's ability to increase signal detection rate, keeping false alarm rate constant

- Develop signal processing algorithms for radar detection within the reprocessing paradigm

These goals were slightly modified as a result of meetings with Rome Lab representatives and the principal investigators in the spring and early fall of 1992. It was decided that significantly more effort should be put into building a demonstrable IPUS radar testbed based on positive preliminary results with the sound understanding testbed. Our investigations of the communication application were thus scaled back and less effort expended in the development of the generic architecture. As will be discussed in detail in the body of this report, we have met or exceeded our revised project goals.

The following highlights the major accomplishments of the project. When we began this effort, there was little understanding by either the AI community or signal processing community of the importance of the IPUS framework. As a result of the efforts on this project, we have been able to implement in detail the architecture and demonstrate its operation in two complex signal processing application domains (radar and sound understanding). This has resulted in the acceptance of IPUS as a major new paradigm for complex signal interpretation tasks as indicated by the acceptance for publication of a paper describing the architecture in the AI Journal, a keynote address on IPUS in Japan, and the publication of over 27 articles describing different aspects of the architecture. Additionally, there were four Ph.D. theses published and there are number of other students in advanced stages of completion.

The major experimental evaluation of IPUS has been in the sound understanding application involving the recognition of multiple, time-overlapping household sounds. This application was built on the IPUS generic testbed implemented in LISP. In developing this application, we have had to deal with issues of real-world data involving significant noise, multiple signal processing algorithms and techniques for the integration of their output, and complex control strategies for handling large libraries of sound sources. In supporting this application, we had to refine and generalize the RESUN architecture which the IPUS generic testbed is built on. This involved the development of canonical evidence combination methods, negative evidence uncertainty in belief summaries, approximate knowledge representation and summarization, and efficient implementation and extension of event-based refocusing conditions.

Based on the IPUS generic testbed architecture and RESUN architectures, we have been able to transfer IPUS technology from a LISP environment to a C++ environment for use in an IPUS Radar Clutter Analysis Testbed. Though not as well developed as the sound understanding application because of its newness, this radar testbed has still clearly demonstrated the potential of IPUS-like technologies for CFAR processing of radar returns. There has also been significant development of knowledge for weak signal detection. This knowledge has involved the application of the Ozturk algorithm to hypothesize the distribution of data in a clutter patch based on a small amount of data. Also, techniques have been developed for partitioning the radar surveillance volume into background noise and clutter patches, for weak signal detection in K-distributed clutter, and the efficient use of Rejection Theorem for Weibull clutter generation. Though the effort on the application of IPUS to the communication was given less priority, we still did some interesting theoretical work on weak signal detection in communication systems subject to SIRP interference.

An unanticipated outgrowth of our work on IPUS has been the development and demonstration of approximate algorithms in signal processing domains and the beginnings of qualitative theory about IPUS usage.

The remainder of this report documents these contributions in detail. The first section gives a detailed description of IPUS architecture and its implementation in the sound understanding domain. As part of this description, we describe an experimental evaluation showing the advantages of the signal reprocessing over a non-reprocessing version of the testbed. The second section discusses the C++ IPUS Radar Clutter Analysis Testbed with examples of how radar processing can be structured to exploit the IPUS architecture. The third section presents the radar knowledge appropriate for use in such a framework. The fourth section details extensions to and generalizations of the RESUN architecture to support the IPUS framework for complex signal domains. As part of this report, we are also including a number of appendices that represent a further level of detailed discussion. Appendix A lists all the publications associated with the project; Appendix B is a copy of the paper on IPUS that will be appearing in early 1995 in the AI Journal; Appendix C is a manual for use of the C++ radar testbed; Appendices D, E, F and G are theses from Syracuse University and represent work on the development of knowledge for the application of IPUS to statistical domains. Appendix D is R. Shah's MS thesis and discusses a new technique for distribution approximation of random data. Appendix E is M. Rangaswamy's Ph.D thesis which discusses spherically invariant random processes for use in radar clutter modeling, simulation and distribution identification. Appendix F is P. Chakravarthi's Ph.D thesis which discusses the issue of subclutter visibility in the context of weak signal detection. Finally, Appendix G is M. Slamani's Ph.D thesis which lays out a new approach to radar detection based on the partitioning and statistical characterization of the surveillance volume.

Chapter 2

Acoustic Interpretation Testbed

2.1 Introduction

This chapter summarizes the evolution of the generic IPUS testbed over the term of the contract. We present the framework's initial formulation and its initial implementation plans in Section 2.2, then describe changes and additions with respect to this formulation in Section 2.3. In Section 2.4 we discuss the design of an evaluation experiment for the acoustic testbed and present the testbed's performance. We summarize our conclusions and future research directions in Section 2.5. Note that Appendix B contains an in-depth journal article on the IPUS architecture.

2.2 Initial IPUS Conception

2.2.1 Framework Philosophy

The IPUS generic testbed is intended as a framework for next-generation signal-interpretation systems. Current systems have most often assumed that fixed signal processing in the front-end can provide adequate (not necessarily optimal) evidence for reliable interpretations regardless of the range of possible scenarios in the environment. In our opinion, this assumption is plausible for systems that monitor stable environments, but not for those that monitor complex environments. In these environments, the choice of front-end signal processing algorithms (SPAs) is crucial to the generation of adequate evidence for interpretation processes. Parameter values inappropriate to the current scenario can lead to a front-end providing misleading data correlates to its system's high-level interpretation components. In such circumstances the system might become unable to interpret entire classes of environmental events correctly. Front-end SPA sets for complex environments must be dynamically modifiable to respond to scenario changes and to reprocess ambiguous or distorted data. "Dynamically modifiable" refers both to the ability to change a particular SPA's control parameter values and to the ability to select entirely new sets of front-end SPAs.

Our design of the framework IPUS concentrated on controlling the interaction between front-end SPAs and highlevel interpretation processes in a signal interpretation system. Specifically, in designing the IPUS paradigm we wanted to develop a framework that would

1. enforce structured, bidirectional interaction between the search for front-ends appropriate to the scenario and the search for interpretations appropriate to data correlates produced by the front-ends.

2. adapt processing strategies to the emerging interpretation of current data and to emerging expectations for future data,

3. organize and apply formal signal processing knowledge to recover from uncertainty introduced by front-end numerical SPAs.

Accordingly, the basic IPUS framework was specified to use an iterative process to converge to appropriate SPAs and interpretations. For each data block, the loop would start by processing the signal with an initial SPA configuration. These SPAs would be selected not only to identify and track the objects most likely to appear, but also to provide indications of when less likely or unknown objects have appeared. In the next loop step, a *discrepancy detection* process would test for discrepancies between the correlates of each SPA in the current configuration and expectations based on (1) object models, (2) the correlates of other SPAs in the configuration, and (3) application-domain signal characteristics. These comparisons could occur both after SPA output is generated and after interpretations are generated. If discrepancies were detected, a *diagnosis* process would then attempt to explain them in terms of a set of distortion hypotheses. This diagnosis would use the formal theory underlying the signal processing. The loop would end with a *signal reprocessing* stage that proposed and executed a search plan to find a new front-end (i.e. a set of SPAs) to eliminate or reduce the hypothesized distortions. After the loop's completion, if there were any similarly-rated competing top-level interpretations, a *differential diagnosis* process would select and execute a reprocessing plan to detect features that discriminated among the alternatives.

Figure 2.1 summarizes the architecture. The dual search in the framework becomes apparent with the following observations. Each time data is reprocessed, whether for disambiguation or distortion elimination, a new state in the SPA search space is tested for how well it eliminates distortions. The measurement of distortion elimination or disambiguation assumes that the system's current state in the interpretation space matches the scenario being observed. Failure to remove a hypothesized distortion after a bounded search in the SPA space will lead to a new search in the interpretation space. This occurs because the diagnosis and reprocessing results represent attempts at justifying the assumption that the current

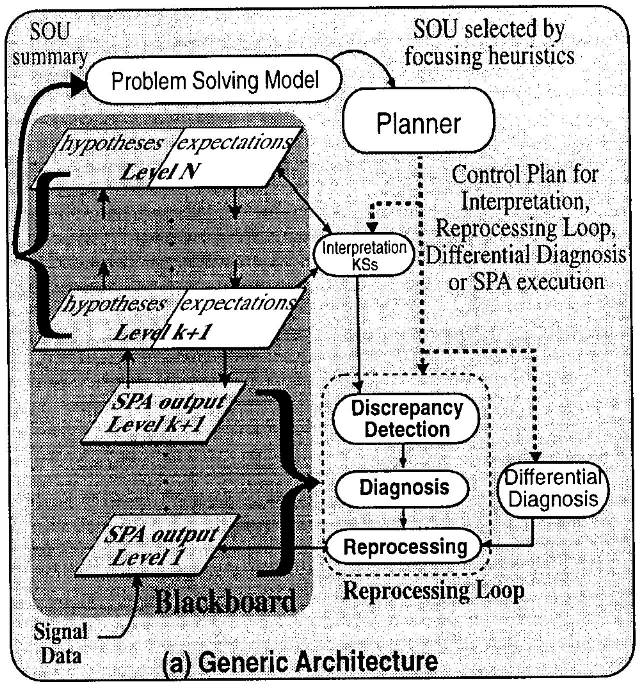


Figure 2.1: *Generic IPUS Architecture*

interpretation is correct. If either diagnosis or reprocessing fails, there is a strong likelihood that the current interpretation is not correct and a new search is required in the interpretation space. Furthermore, the results of failed reprocessing can constrain the new interpretation search by eliminating from consideration objects with features that should have been found during the reprocessing.

We intended IPUS-based systems to be able to manage the uncertainty levels of their interpretations. Therefore, we had to provide the architecture's control framework with a way to represent factors that affect interpretations' certainties. The control framework also had to support context-sensitive focusing on particular uncertainties in order to control engagement and interruption of the architecture's reprocessing loop.

For these reasons, IPUS uses the RESUN framework to control knowledge source (KS) execution. This framework supports the view of interpretation as a process of gathering evidence to resolve hypotheses' sources of uncertainty (SOUs). It incorporates a language for representing SOUs as structures which trigger the selection of appropriate interpretation strategies. Problem-solving is driven by information in the *problem solving model*, which is a summary of the current interpretations and the SOUs associated with each interpretation's supporting hypotheses. An incremental, reactive planner maintains control using *control plans* and *focusing heuristics.* Control plans are schemas that define the strategies available to the system for processing and interpreting data, and for resolving interpretation uncertainties. Focusing heuristics are context-sensitive tests to select SOUs to resolve and processing strategies to pursue.

2.2.2 Testbed Implementation

The IPUS-framework testbed was initially implemented as follows. A blackboard-based approach was used for storing and maintaining interpretation hypotheses and SPA results, though the traditional agenda-based balackboard control was replaced by RESUN. Front-end SPAs were specified within the RESUN control-plan language simply as parameterized primitive plans. When there was more than one possible SPA to apply (e.g., three available peak-picking SPAs), focusing heuristics consulted a system variable that determined which SPA was considered part of the current front-end, and with which parameter values it would be executed.

The discrepancy detection component was implemented as a set of specialized KSs, each of which was executed after its particular front-end SPA had been applied. Based on the results of their discrepancy-detection tests, these KSs attached appropriate SOUs to SPA outputs. The SOU would point to all outputs concerned in the discrepancy. SOUs related to conflicts between data and model-based expectations were generated by high-level interpretation KSs whenever the KSs failed to find data to support models' features. The discrepancy diagnosis component was implemented as a control plan that was executed when particular discrepancy-related SOUs occurred in the RESUN problem-solving model. It examined the hypotheses linked with discrepancy-related SOUs and generated a list of distortion diagnosis operators as an explanation for the discrepancy. The reprocessing component was implemented as a single control plan whose reprocessing subgoals were solved by other plans selected by focusing heuristics according to the distortion explanation. The differential diagnosis component was implemented similarly to the discrepancy diagnosis component.

Figure 2.2 shows the initial seven abstraction levels used by the testbed to represent signal data and acoustic interpretations. The lowest level was the segment level. A segment is a collection of waveform points to which some SPA will be applied. Time-domain statistics such as zero-crossing density, average energy, etc, are also maintained for segments. The second level consisted of spectral hypotheses derived for each segment through Fourier-Transform-based algorithms such as the STFT and Wigner-Distribution algorithms. The third level consisted of peak hypotheses derived for each spectrum. The fourth level consisted of contour hypotheses, which are group of peaks whose time indices, frequencies, and amplitudes represent a contour in the time-frequency-energy space with uniform frequency and energy behavior. The fifth level contained microstream hypotheses supported by a sequence of one or more contours. Each microstream has an energy pattern consisting of an attack region (signal onset), a steady region, and a decay (signal fadeout) region. Groups of microstreams synchronized according to time and/or other psychoacoustic criteria such as harmonic frequency sets supported stream hypotheses in the sixth level. At the seventh level, sequences of stream hypotheses were interpreted as sound-source hypotheses.

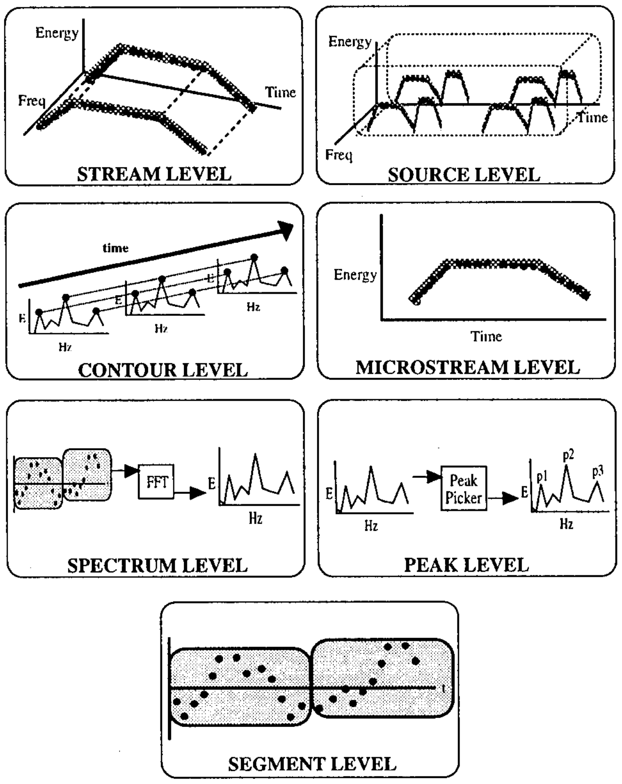


Figure 2.2: *Original Testbed Evidence Abstractions.*

To support the posting of SOUs about behavioral expectations for sources, a grammar was used to represent the various kinds of stream combinations that could occur while a source is active. Three classes of stream combination were recognized and represented accordingly:

- sequential events: (:SEQ *exp*1 *exp*2 ... *exp*N)
- simultaneous events: (:PARALLEL *exp*1 *exp*2 ... *exp*N)
- periodic events: (:ITER (min-iterations max-iterations) *exp*)

where *exp* is a single event or another combination.

The initial testbed's interpretation control strategy was strongly data-driven. For each data block the front-end SPAs were applied to ultimately produce contours. The contours were then used to hypothesize microstreams, which in turn were used to partially specify streams. These partial streams were used to generate source hypotheses. Verification of a source hypothsis consisted of finding contours that would support the source's remaining unsupported microstreams.

2.3 IPUS Framework Refinements and Extensions

Several shortcomings in the original framework's formulation became apparent in the course of applying the testbed to the acoustic signal interpretation problem. The original IPUS architecture specification did not include the concept of re-use of reprocessing results, nor did it have a formal mechanism for fusing results from executions of SPAs with different parameter values. The process of adding SPAs to the testbed was found to be tedious and error-prone because it "spread out" SPA knowledge throughout the testbed files, instead of organizing it in one SPA declaration. We also found it necessary to modify the signal abstraction hierarchy and add approximate reasoning techniques to accomodate realworld sounds more effectively. The changes introduced by these shortcomings resulted in the architecture depicted in Figure 2.3. The next three subsections describe our approaches to the above problems.

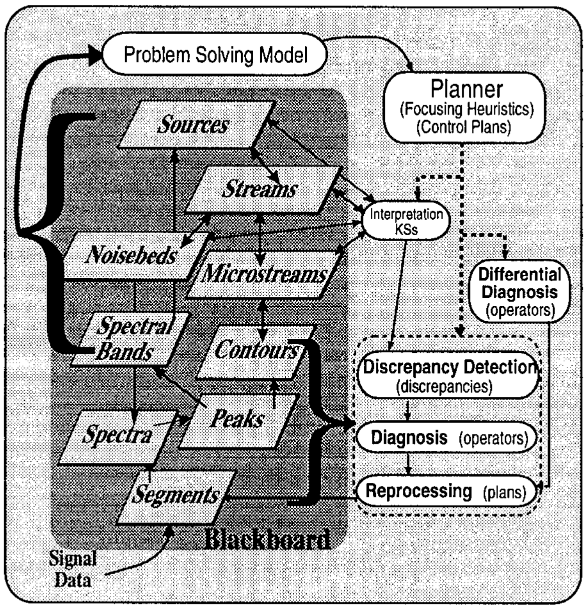


Figure 2.3: *Current IPUS Acoustic Testbed.*

2.3.1 Processing Contexts and SPA Information

In regard to the reprocessing, SPA-application, and data fusion issues, we found it necessary to extend the formal definition of SPAs to include the concept of *conflict discrepancy-detection tests* and to extend the generic framework's system code to generate and manipulate data structures for representing *processing contexts*, and *context-mapping rules.*

Conflict discrepancy-detection tests are a set of comparisons $\mathcal{C}$ defined on $\mathcal{F} \times \mathcal{F}$, where $\mathcal{F}$ is the set of generic front-end SPAs available to the IPUS system. Each test represents a heuristic consistency check that can be made between correlates of one generic SPA's instances and correlates from another generic SPA's instances. These tests are based on signal processing theory and are used to indicate when one SPA instance might no longer be capturing *all* desired features from the environment . For example, one test from the IPUS acoustic testbed compares the energy fluctuations from a time-domain energy tracker with the appearance of new peak tracks in the output of an STFT. It is theoretically valid in the sense that it is based on the requirement by formal processing theory that time-domain signal energy must be conserved in the frequency-domain. It is heuristic in the sense that not all time-domain energy "bursts" are due to the appearance of new sounds; sometimes they arise from interactions among the currently-active sources.

A processing context is a structure that stores relevant assumptions made by the IPUS system at the time a correlate was produced. In IPUS, every SPA correlate is tagged with a processing context. Specifically, the processing context contains:

1. the correlate's parameter context. This is the generic SPA whose instance actually produced the correlate and the values the SPA instance's parameters had at the time the correlate was produced.

2. the correlate's processing history. This is the chain of SPAs (with parameter values) that traces the creation of the correlate from the raw signal waveform.

3. the problem-solving goals in effect when the correlate were produced. For example, the goal of reducing uncertainty resulting from alternative interpretations for the same data, or the goal of finding correlates for a particular frequency track of a particular source.

4. the time period(s) for which the context is true.

By themselves, processing contexts are useful for model synthesis in that the provide a history of how the environment signal was processed, and with what results. Thus, they can support efficient re-use of previous reprocessings' results. By including data-checks in the definitions of each reprocessing plan's subgoals, we made the testbed's reprocessing component capable of re-using previous results. By examining in these data-checks the parameter values and SPAs of previously-created processing contexts, the IPUS reprocessing component can avoid re-execution of any previously-executed reprocessing plan steps. In such cases the existing SPA results are retrieved and passed to the next plan step. In addition to strict parameter-value equality tests, the data-checks can also permit, when appropriate, re-use of correlates from processing contexts with tighter parameter values than those in the current proposed plan.

Processing contexts are most useful for model synthesis, however, in connection with context-mapping rules. Context-mapping rules are transformations defined on processing contexts. Each transform maps correlates computed by one instance of a generic SPA to their expected appearance if computed by a different instance of the SPA. The input to the transforms consists of observed correlates and their processing contexts, and the output is what correlates should appear under the new parameter context *assuming the other information in the original processing context is valid.* These transforms are useful in the IPUS reprocessing component in situations where correlates from two processings of the same signal must be compared to find evidence for a new object that is discernible only in the second processing's correlates. By mapping the context of the first processing into that of the second processing, we can eliminate from consideration those correlates in the second processing that correspond to previously-identified correlates in the first processing.

In the latest IPUS acoustic interpretation testbed, SPA definitions have been extended to require hand-crafted context-mapping rules and discrepancy detection tests in addition to the code for the SPA proper. For example, the mapping rule for the STFT covers feature changes including the merging and splitting of peaks as frequency and/or temporal resolution changes and the magnitude fluctuations that peaks can undergo due to beat phenomena. The peak-picker SPAs' discrepancy detection tests compare new peaks' energy with fluctuations in time-domain SPAs' outputs.

2.3.2 Approximate Processing

Another testbed shortcoming was found when the testbed was applied to realworld sounds. The original sound understanding testbed was tested on synthetic narrow-band signals. When we tried to run realworld sounds through the testbed, testbed recognition rates suffered. The reason is that most real sources don't have clean frequency components. This can be because of wideband source components or because a source has closely- spaced harmonics. In either case, bottom-up spectral analysis of realworld acoustic scenarios produces band-limited peak "clouds" instead of clean peak tracks that can be unambiguously contoured. Effective bottom-up contouring is not possible in these situations. It is impossible to say which contours are present unless the contouring procedure is expectation-driven. Any other type of contouring would be misleading. Thus, we found it necessary to implement a new control strategy (called configuration /em C.2 and two new abstractions: noisebeds and spectral bands. We postpone discussion of noisebeds since we will use them as an example of how new knowledge can be added in a structured manner to IPUS-based systems.

Spectral bands are supported by clusters of peaks detected in spectral analysis. They are hypotheses formed to indicate frequency-time regions of uniform spectral activity without regard to the narrowband constraints of contouring. As will be seen in the description of the new control strategy, the spectral bands' creation and subsequent analysis are examples of *approximate processing*, where only a rough estimate of data characteristics is produced first and later incrementally refined.

The use of approximate knowledge allows us to have an early rough global view of the possible sources present in the scenario as oppose to the partial "exact" view of some of them provided by the original bottom-up contouring approach. Because of its rough-global-estimate nature, this view provides better support for differential diagnosis. A differential diagnosis process is most useful for discriminating among several similarly-rated rough hypotheses as opposed to discriminating between two differently-rated precise hypotheses. By the generation of early source expectations based on approximate knowledge, reprocessing needs may be detected right away, as soon as a source is selected for verification.

A summarized description of the new control strategy loop is:

1. apply SPAs to the input waveform (i.e. spectral analysis culminating in peak hypotheses generated by the peak-picker SPA from spectra produced by a Short-Time Fourier Transform SPA) and obtain spectral bands.

2. identify a small group of possible sources based on all their components' overlaps with all observed bands.

3. generate expectations for the components of the hypothesized sources.

4. do focused contouring in narrowband regions of hypothesized sources to verify the component expectations.

5. verify the component expectations.

Whenever a contour is found to support a component, this new information about the source is propagated to the spectral band. This produces a refinement (narrowing, or at the least, removal of peaks) of the spectral bands, and recomputation of the explanation inferences associated with them. As result of this computation, some previously supported sources may not be supported any longer and some parts of the spectral bands may be labelled as *unexplained*, reflecting the fact that we missed some explanations (sources present in the scenario) in the initial approximate processing. One of the advantages of using approximate knowledge is a more focused search in the library of possible source models. Without approximate knowledge, the access to the library is based on single frequency components, because without expectations there is no way to join partial microstreams into streams. Many sources would map into a single frequency track. Each of these sources would have to be verified. However, with approximate knowledge the source library is indexed with all the frequency bands with energy in the scenario, obtaining the set of sources that map such a frequency picture. Top down "focused" contouring as oppose to "blind" bottom-up contouring can then be used.

2.3.3 IPUS Knowledge Engineering

As we added new knowledge to the acoustic interpretation testbed, we realized our architectural goal of organizing signal processing theory for handling front-end correlates' uncertainty would require a formal methodology for adding signal processing knowledge to IPUS-based systems.

We therefore developed a domain-independent framework that classifies signal processing knowledge into five classes:

1. signal representation: knowledge in the form of concepts used to symbolically abstract signals. (e.g., data structures to peaks in a spectrum produced by a Fourier Transform algorithm)

2. SPA/interpretation: knowledge in the form of signal processing algorithms or interpretation KSs. (e.g., varieties of peak-detection algorithms and how changes in their control parameters' values influence their identification of peaks, or methods for combining frequency components into streams.)

3. discrepancy detection check: knowledge in the form of methods for making formal comparisons between an SPA's correlates and (1) other SPAs correlates, (2) environmental constraints, and (3) model-based expectations. (e.g. code for comparing time-domain signal energy with energy of frequency tracks in the output of a series of FFTs)

4. diagnosis distortion modeling: knowledge in the form of distortion processes that can be introduced into an SPA's correlates due to inappropriate parameter settings. These distortion models are used by the diagnosis component to explain discrepancies. (e.g. Assume that a short-time Fourier Transform (STFT) with an analysis window of length W is applied to a signal sampled at rate R. If the signal came from a scenario containing frequency tracks closer than R/W, Fourier theory predicts that the tracks will be merged in the STFT's computed correlates.)

5. reprocessing strategy: knowledge in the form of generic processing strategies for removing distortions or disambiguating between competing alternative interpretations of a signal. (e.g. If a short-time Fourier Transform's (STFT) correlates are hypothesized to suffer from merged peaks due to low frequency resolution, then a generic strategy of gradually increasing the STFT's analysis-window parameter should reduce or remove the distortion.)

The classes form a hierarchy. The addition of knowledge in a class with rank R will require the addition or adjustment of knowledge from the classes ranked after R. When users want to add knowledge to an IPUS-based system, they first identify the class of knowledge they plan to add, then use the hierarchy to identify additional knowledge to be added or maintained. Finally, all the new knowledge is added in the order of the involved classes' ranks. Adherence to this order greatly facilitates the incremental debugging and testing of the new knowledge.

Examination of a wide range of domains reveals two generic classes of correlates: *point correlates* and *region correlates.* A point correlate is a value associated with one point in the SPA output coordinate space. A region correlate is a value associated with a subset of the SPA output space. A spectral peak energy value in the "time, frequency, energy" space of acoustic signal processing is an example of a point correlate. A noise-distribution tag for a region in a radar sweep is an example of a region correlate. A track of spectral peaks over time from a series of FFT analyses is an example of a region correlate comprised of non-contiguous subsets of the SPAs' output space.

We further subdivide the "discrepancy detection check" knowledge into tests for categories of discrepancies. The IPUS discrepancy-detection component's formal specification has been updated to require that discrepancy tests be classed as checks for one of the following generic discrepancies between an SPA's anticipated correlate set and its computed correlate set. The different domains of the examples we provide for each discrepancy are intended to indicate the discrepancy classes' generality.

- missing: An anticipated correlate is not in the computed correlate set. An example of this discrepancy in the acoustic domain occurs when a spectral peak is expected in the output of an FFT, but is not found.

- unassociated: An unanticipated correlate occurs in the computed correlate set. An example of this discrepancy in the radar domain occurs when an unanticipated clutter region is produced during a radar sweep.

- value-shift: A correlate is found in the computed correlate set at its anticipated coordinates, but with an unanticipated value. In the visual domain we encounter this discrepancy when an image region's hue label produced by an intensity analysis SPA is brighter than expected.

- coordinate-shift: A correlate with an anticipated value is found in the computed correlate set but at unanticipated coordinates. This includes the situation where a region's boundaries shift from their expected locations. An example of this discrepancy in the acoustic domain occurs when a track of spectral peaks produced by a curve-fitting algorithm has the correct energy value but is 30 Hz from its expected position.

- merge: Two or more anticipated correlates are deemed to have appeared as one unanticipated correlate in the computed correlate set. The criteria for this merging are domain-specific and often depend on relationships between the missing correlates' values or coordinates and the unanticipated correlate's value or coordinates. An example of this discrepancy in the visual domain occurs when two adjacent regions with different expected textures are replaced by one region with an unanticipated texture.

- fragmentation: An anticipated correlate is deemed to have been replaced by several unanticipated correlates in the computed correlate set. The criteria for this splitting are domain-specific and often depend on relationships between the missing correlate's values or coordinates and the unanticipated correlates' values or coordinates. An example of this discrepancy in the radar domain occurs when a noise-analysis SPA computes two or more small regions with a particular noise-distribution label instead of an expected single region with that label.

For an example of how this knowledge engineering methodology works, consider the process in which we added the noisebed abstraction to the acoustic interpretation testbed. Noisebeds were intended to model wideband-energy spectral characteristics of realworld sources, and would be supported by a set of spectra on the basis of how closely the spectra features matched the noisebeds' feature-vectors. In their turn, noisebeds would serve as support for stream hypotheses of the realworld sounds.

There four features used in the noisebed feature-vector. They are all generated from an 8×10 tiling of the first 0.2 seconds in the frequency$\times$time spectral analysis during which the noisebed's sound source is suspected to be present. The noisebed features are defined as

1. Center of gravity (i.e. average of frequency indices weighted by the spectral energy found in the indices' tiles) in the time slice containing the tile with maximum spectral energy over the entire tiling.

2. Difference in energy between the maximum-energy tile and the tile with the same frequency index and the next-higher time index.

3. Difference in energy between the maximum energy tile and the tile with the same time index and the next-higher frequency index.

4. Difference in energy between the maximum energy tile and the tile with the same frequency index and the next-lower time index.

Following our knowledge engineering scheme, we started to incorporate noisebeds into the testbed by first adding the noisebed abstraction. This involved defining blackboard units and accessor functions, and updating the source-model specification language to include the noisebed concept.

The next step in the knowledge hierarchy, SPA/interpretation knowledge, required us to specify and implement 1) KSs to group spectra and generate appropriately-initialized (e.g., correct SOUs, hypothesis-slot values, and feature values) noisebed hypotheses from them, 2) KSs to include noisebeds as support for streams, 3) functions to maintain the blackboard noisebed hypotheses' slot values for cases when new evidence for the noisebed's source is found or when the noisebed's source is disbelieved, and 4) evidence combination functions for rating streams on the basis of their noisebeds' sous.

To incorporate the noisebed-related knowledge from the third knowledge category, discrepancy checking, we added control plans that generated, whenever a new stream was hypothesized in a top-down manner, noisbed expectations in addition to the stream's expected microstreams.

The distortion modelling class of knowledge required us to specify in focusing heuristics which SOUs on a noisebed hypothesis or in the problem solving model would trigger the discrepancy diagnosis control plan. We also had to add distortion operators to the diagnosis KS's operator database to describe the distortions we expected to occur with noisebeds. This included the noisebed-summing and noisebed-masking operators. The first applied to situations where the added energy of an underlying noisebed would raise the observed energy of a microstream beyond its expected bounds. The second applied to situations where a low-energy microstream would be overwhelmed (masked) by the higher energy of a surrounding noisebed.

The process of incorporating noisebeds into the IPUS framework culminated in identifying and adding knowledge from the reprocessing strategy knowledge class. This involved adding 1) differential diagnosis tests to avoid specifying goals that would require search for noisebed-masked features, 2) narrowband-filtering control plans, 3) wideband-filtering control plans. The second set of plans would be executed to eliminate narrowband microstream tracks from other sources that overlapped a noisebed just before reprocessed spectra were statistically analyzed to confirm the noisebed's existence. This was necessary since a source's noisebed statistical models were generated for the source in isolation; any extraneous tracks would unfavorably skew the statistical measurements. The third set of control plans would be used during reprocessing to remove noisebeds before searching for masked microstreams.

2.4 Performance Analysis

For an indication of the reprocessing architecture's usefulness in the acoustic domain, this section presents an experiment comparing the testbed's performance on a set of 30 scenario signals randomly generated from a library of 5 realworld sounds, *with* and *without* the reprocessing capability. In both cases, the same initial front-end SPAs and parameter settings were used throughout the suite.

Figure 2.4 shows the library sounds, and Table 2.1 shows the initial parameter settings of SPAs that could be used in reprocessing. The STFT parameter settings were selected to provide adequate time resolution to detect the `Siren-Chirp`'s attack, and the peak-picking parameters were selected to suppress low-energy noise peaks along the chirp tracks that could interfere with accurate contouring of the chirp's peaks.

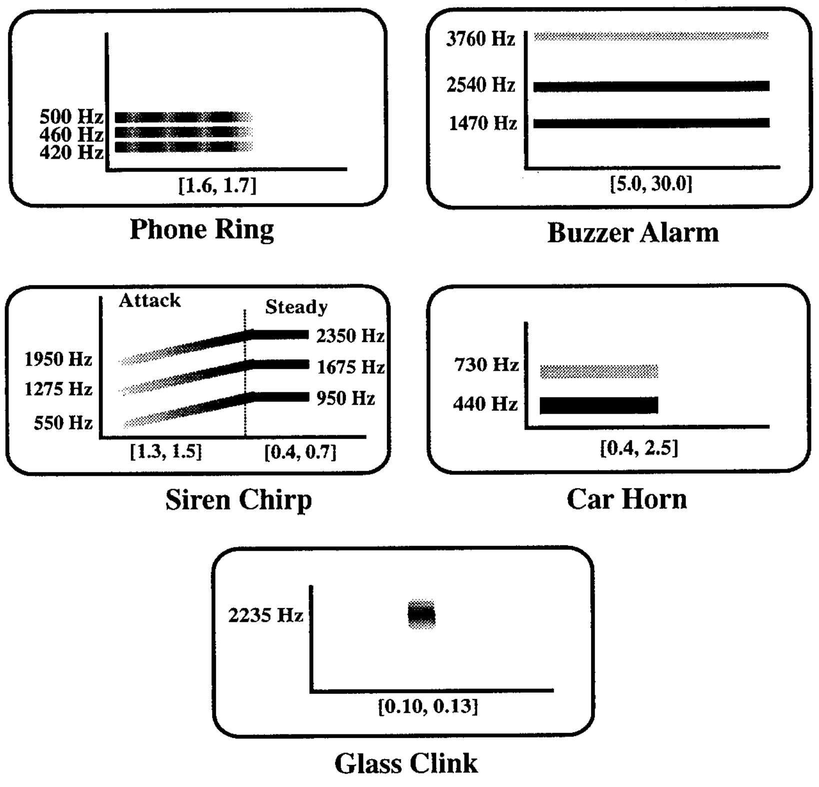


Figure 2.4: *Library of real-world sounds used to generate the experiment's scenario suite.*

The 30 scenarios were generated according to the following protocol. For each scenario, the `Burglar-Alarm`, `Siren-Chirp`, and `Phone-Ring` had a 100% chance of occurring, and had their start-times uniformly distributed over the time range [0.04.0] seconds. These three sources' durations were selected for each scenario from a uniform distribution over the time ranges specified in Figure 2.4. The last two library sources, `Car-Horn` and `Glass-Clink`, had a 50% chance of occurring in each scenario. This reduced occurrence rate was used so that the scenario test suite would give an indication of the false-alarm rate for sources that could either be confused with others (i.e. `Car-Horn` and `Phone-Ring`) or could be overwhelmed by other sources that overlapped in time and frequency (i.e. `Glass-Clink`). Over all scenarios, the relative source energies of `Burglar-Alarm`, `Phone-Ring`, and `Car-Horn` were kept at a 1.0:1.0:1.0 ratio, while `Siren-Chirp` was 1.2 times and `Glass-Clink` was 0.8 times as energetic as those three sources. Each scenario ended when its last source turned off. The minimum generated scenario length was 5.2 seconds, the maximum was 8.4 seconds.

The testbed's distortion operator database had 6 distortion operators, and the testbed had reprocessing plans available for every conceivable combination of these distortions in a diagnosis explanation. The distortion operators and the distortions they modelled were:

SPA	Parameters	Initial Value
Peak-Picker	PEAK-NEIGHBORHOOD	2
	NOISE-PEAK-THRESHOLD	0.04
	PEAK-LIMIT-PER-SPECTRUM	6
Short-Time Fourier Transform (STFT)	ANALYSIS-WINDOW	256
	FFT-SIZE	512
	DECIMATION	256

Table 2.1: *Front-end SPAs and their initial parameter settings.* `PEAK-NEIGHBORHOOD` *is the number of fourier-transform energy sample-points on both sides of a candidate peak energy value that the value must be greater than in order to be considered a peak.* `NOISE-PEAK-THRESHOLD` *is the absolute minimimum energy value a fourier-transform sample-point must have in order to be considered a peak.* `PEAK-LIMIT-PER-SPECTRUM` *is the maximum number of peaks that can be returned from a single spectrum. DECIMATION is the separation between consecutive analysis window positions.*

- ENERGY-THRESHOLDING: a microstream's support was missing because the peak-picker SPA's energy threshold was too high.
- PEAK-LIMITATION: a microstream's support was missing because the peak-picker SPA's peak limit was too low and the number of other microstreams was greater than the peak limit.
- WIDE-PEAK-LOCALITY: a microstream's support was missing or only partially observed because the peak-picker SPA's peak neighborhood was too wide; moderate-energy peaks were overwhelmed by the energy of nearby high-energy peaks.
- FREQUENCY-RESOLUTION: the STFT's analysis window was too short to generate spectra with adequate frequency resolution for discerning peaks for closely-spaced (or overlapping) microstreams.
- BINWIDTH-RESOLUTION: the STFT's FFT length was too low to provide enough sample points of the spectrum for discerning the existence of peaks.
- TIME-RESOLUTION: the STFT's decimation was too high to accurately track time-dependent changes in microstream frequencies or energies.

For each scenario we compute a B-vector (Belief-vector) to measure the testbed's success at interpreting the scenario signal and the total time the testbed took to process the scenario. The B-vector contains two numbers, a weighted average belief for correctly-identified sources C and a weighted total belief for false-alarms F. These two values are defined as follows:

$$C = \sum_{i=1}^{N} b_i(d_i/D_i)/N, \tag{2.1}$$

$$F = \sum_{j=1}^{M} b_j(d_j/D_j^e), \tag{2.2}$$

where N is the number of sources actually in the scenario and M is the number of false-alarm source identifications. The value b_i is the testbed's overall belief value associated with the i-th source, while d_i is the duration for which the i-th source was tracked in the scenario, D_i is the actual duration of the source in the scenario, and D_i^e is the average duration length for the i-th source in the source database.

2.4.1 Experiment Expectations

The reprocessing capability was disabled by simply specifying no action for the SOUs that normally would trigger diagnosis and reprocessing. Because these SOUs are negative evidence against the existence of a source, we expect the non-rerocessing experiment runs to have lower average beliefs than those of the reprocessing experiment runs on the same scenarios. Reprocessing can possibly lead to "hallucinations," or situations where poor data is forced to support a source because a diagnostic explanation requires it. However, reprocessing can also lead to the resolution of ambiguities in source identification or more accurate source time-bounds. For these two reasons, there is no clear expectation for comparisons between false-alarm belief values between the two testbed versions.

2.4.2 Experiment Results

The scenario experiments were performed on a TI Explorer II+ with garbage collection disabled. All SPAs were implemented in LISP with error-checking code intact, hence FFT-based algorithms such as the STFT greatly increased the actual execution times over real-time constraints. The average runtime for scenarios without reprocessing was 568 seconds, whereas the average runtime for the same scenarios with reprocessing was 655 seconds. On average, the reprocessing architecture required 15.3% more time than the fixed-front-end approach. However, the average B-vector for reprocessing was $< 0.67, 0.1 >$ while the average B-vector for the fixed front-end was $< 0.48, 0.19 >$, indicating that the reprocessing architecture generated more credible results and a lower belief in false-alarms than the fixed front-end.

The false-alarms for the fixed front-end occurred primarily when spectral energy in the 400–500 Hz region was confused between `Phone-Ring` and `Car-Horn` when only one of the two were present. These errors were usually corrected in the reprocessing testbed's reprocessing search for `Car-Horn`'s 700 Hz track or in the testbed's reprocessing search in the 400–500 Hz range with greater STFT settings for greater frequency resolution. The reprocessing testbed's false-alarms occurred primarily when time-domain energy fluctions due to sources turning on or off at times separated by less than 0.3 seconds were confused with the `Glass-Clink` source.

The higher average belief in successful recognitions for the reprocessing testbed can be attributed primarily to tighter bounds on the sources' start- and end-times attained by reprocessing moderately-wide time-regions where the endpoints were suspected. A secondary reason for the higher belief is due to the reprocessing testbed's searches for low-energy source-components (microstreams) that were lost due to energy-thresholding.

2.5 Summary

Our work in applying the IPUS architecture to the problem of interpreting real-world acoustic scenarios had to deal with issues of real-world data involving significant noise, multiple signal processing algorithms and techniques for the integration of their output, and complex control strategies for handling overlapping and similar sounds.

This emphasis on real-world usage resulted in three general extensions to the architecture:

- a "processing context"-based framework to support the integration of output from signal processing algorithms with different parameter settings,
- the addition of approximate-processing techniques to constrain combinatorial search for interpretations of initial, bottom-up front-end processing results, and
- a theory of knowledge engineering for structuring the addition of signal processing knowledge to IPUS-based systems.

Our work has also resulted in an initial evaluation process based on B-vectors for analyzing the utility of reprocessing and the importance of modelling various distortions in diagnostic distortion operators.

There are two categories of topics for future investigation in the IPUS project: domain-independent architectural enhancements or software tools, and acoustic-domain modelling approaches. The first category includes:

- development of a framework for estimating the amount of reprocessing required in an environment, given the available SPAs and the occurrence rates of objects in the environment;
- specification of a domain-indendent language of higher level than LISP for expressing RESUN focusing-heuristic tests;

- development of a framework for identifying which possible data correlate distortions should be modelled.

The second category of future-work topics includes:

- expansion of the acoustic evidence hierarchy to include expectation knowledge from higher levels than sequential streams, such as acoustic scripts;
- evaluation of various psychoacoustic theories of acoustic streaming (e.g. the grouping of energies at different frequencies and times into patterns representing sources);
- determination of the testbed's performance as the acoustic database grows from tens to hundreds of sound models;
- identification and development of more approximate processing signal processing algorithms for streaming.

Chapter 3

Radar Clutter Analysis Testbed

3.1 Introduction

A major issue addressed in our research concerns the applicability of the IPUS architecture to problems requiring sophisticated processing and interpretation of radar signals. This investigation was motivated by the fact that radar signals are highly non-stationary and therefore require sophisticated interleaving between signal processing and signal interpretation. Our exploration was carried out through the development of a IPUS software system for the radar application. This system was implemented in a C++ environment in order to facilitate the numerically intensive computations associated with radar signal processing.[1] The implementation of IPUS in a different computer language (C++ instead of LISP) also provided us the opportunity to assess the architecture's portability, and to examine associated software engineering issues. Our success in achieving these various goals is reflected by the following major accomplishments:

- Development of a C++ platform for creating IPUS-based application systems. The platform is accompanied by a detailed user manual. (See Appendix C)
- Implementation of a radar clutter analysis testbed (CAT) on the C++ IPUS platform. The CAT system utilizes the radar knowledge developed at Syracuse University.
- Implementation of an X-Windows based user interface for the radar application. The interface can be used to display the contents of the blackboard, the problem solving model and a list of triggered knowledge sources.

On the basis of these accomplishments, which we shall shortly discuss in detail, we have concluded that the applicability of IPUS to radar problems has been success-

[1]The original LISP based platform for the Sound Understanding Testbed was inefficient with respect to numerically intensive computation.

fully demonstrated. In particular, the results obtained justify further exploration of radar problems using the IPUS tools developed in our research.

In order to gain further insight into the design of the C++ IPUS platform, the CAT system, and the user interface, it is necessary to have a certain degree of familiarity with the basics of radar clutter analysis and with the fundamentals of the IPUS architecture. An outline of this background information is presented in Section 2 for clutter analysis and in Section 3 for the IPUS related material. In Section 4, we describe the C++ IPUS platform. Our description of the CAT System is given in Section 5. The user interface is described in Section 6. Finally, in Section 7 we present a summary of our conclusions regarding the potential benefits of the three major software tools developed in the course of our radar-related IPUS research.

3.2 Background on Radar Problem

The IPUS-based CAT system was developed for experimentation with the knowledge-based analysis of data received by a non-imaging radar [4]. The operation of such a radar involves the transmission of pulses and the reception of pulse reflections for the localization of objects in the environment being scanned. This localization may, for example, be performed with respect to a partitioned range-azimuth plane. Each partitioned portion of the plane is referred to as a *cell*. The goal of the entire radar system is to detect targets of interest to within the resolution of a cell. Reflected pulses are processed by the front-end electronics of the radar to assign energy values (called *returns*) to the various cells. Each return always has a contribution due to the electronic noise of the radar. Furthermore, it may have contributions due to reflections from objects which fall within the range-azimuth region defined by the cell. These objects are generally classified into two categories: target and background. Each target (such as an airplane) is assumed to be smaller than the resolution of an individual cell; the reflection from a target is said to give rise to the *target* component of a return. Each background object (a mountain, a forest, a city skyline, etc.) is assumed to be large enough to cover a multitude of contiguous cells; the reflections from such an object are said to give rise to the *clutter* components of the returns. All the contiguous cells whose returns are affected by the same background object (with uniform reflection properties) are said to constitute a clutter subregion. If the reflection properties of the background object are essentially the same throughout its spatial extent, the corresponding clutter subregion is said to be *homogeneous*. However, one also encounters non-stationary characteristics in the form of inhomogeneities within a clutter region. The objective of the CAT system is to analyze a given set of range-azimuth returns in order to:

- estimate the boundary of each homogeneous subregion of clutter.

- characterize the statistical distribution of the returns in each homogeneous subregion of clutter.

As shown by researchers at Syracuse University [5], the above tasks require sophisticated use of a variety of signal processing algorithms. In particular, it appears that the IPUS strategies of discrepancy detection, discrepancy diagnosis and signal reprocessing provide a suitable framework for accomplishing the clutter analysis task. Indeed, as we shall illustrate in later sections, we have successfully incorporated into the IPUS-based CAT system the algorithms developed at Syracuse University for dealing with various aspects of the clutter analysis problem. In order to examine further details of the CAT system, it is necessary to be familiar with IPUS concepts outlined in the next section.

3.3 Background on IPUS

Familiarity with some fundamental IPUS concepts is necessary for appreciating the discussions in later sections on the C++ IPUS platform and the CAT system. A brief outline of these IPUS concepts is presented in this section. More detailed discussions may be found elsewhere [1].

The IPUS architecture provides a sophisticated framework for interleaving signal processing with search and inference processes. The problem solving in an IPUS-based system takes place on a data blackboard which is operated upon by a variety of so-called *knowledge sources* (KSs). The blackboard is organized into a hierarchy of data levels, each designed to hold hypotheses of a particular type generated during the problem solving process. Each knowledge source is a computer program designed to carry out a certain type of processing, provided the blackboard contains data which satisfies "triggering conditions" relevant to that KS. Examples of KSs include signal processing algorithms as well as various procedures for performing search and inferencing tasks. The invocation of KSs takes place at the behest of a control sub-system, which consequently serves as the primary means for accomplishing the interleaving of different tasks. In the IPUS approach to the analysis of non-stationary signals, the tasks to be interleaved fall into eight broad categories (see Figure 3.1). Some KSs are reserved for the initial processing of signal data when it first arrives in the system. The pattern analysis task refers to the extraction of syntactic patterns in the output data from signal processing KSs. If a particular data pattern partially matches one of the stored models, it may then be necessary to search for further evidence to support the retrieved model. In some situations this can give rise to the need for using a different algorithm to carry out a mapping that was performed during the initial processing of the signal data. Yet another role for signal reprocessing arises in the context of the discrepancy detection task associated with the IPUS approach. In its most general form discrepancy detection refers to the search for inconsistencies between data placed on the blackboard by

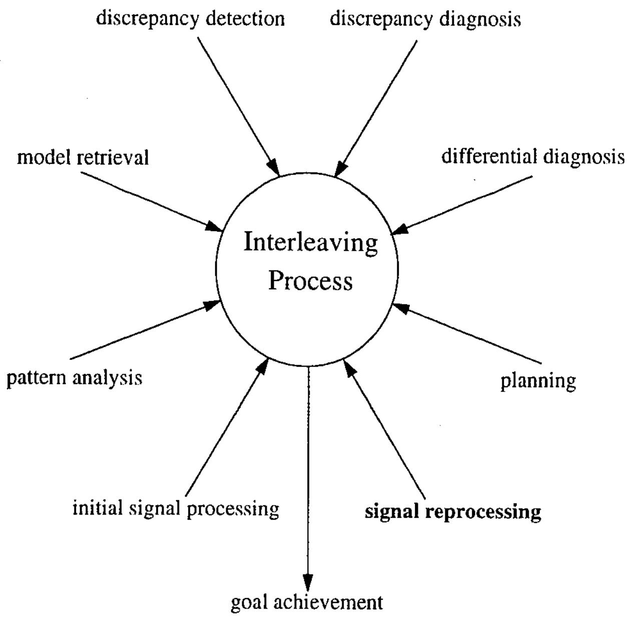


Figure 3.1: Interleaving of different categories of tasks in the IPUS approach for integrated processing and understanding of signals.

different KSs. The diagnosis task in the IPUS approach utilizes the knowledge of signal processing theory in order to diagnose the causes for any detected discrepancies. Consequently, the system may propose a reprocessing plan for the appropriate analysis of the non-stationary signal. The execution of such a reprocessing plan is yet another example of the role that signal reprocessing can play in knowledge-based systems with embedded signal processing.

When formulating signal processing algorithms to be embedded in a knowledge-based system, a signal processing expert also needs to devise control strategies for achieving application-specific objectives with the aid of those algorithms. Such strategies have to specify how to use KSs for signal processing and other tasks in order to achieve specific objectives which may arise in the given application. In the IPUS architecture, such strategies are specified through the RESUN control framework[2] [2]. This framework permits the specification of a particular control strategy in terms of a hierarchy of subgoals interleaved with the actions necessary for achieving them. The specification of the actions necessary to pursue a particular subgoal may include the generation of lower level subgoals as well as the utilization of specific KSs for processing blackboard data. The generation of subgoals may include a specification for carrying them out in a particular sequential order, and conditions for pursuing certain subgoals.

[2]RESUN stands for REsolving Sources of UNcertainty.

As shown in this section, the IPUS architecture is suitable for the analysis of non-stationary signals. In later sections we illustrate this idea through the analysis of non-stationary radar signals using the IPUS-based CAT system. In order to examine the details of the CAT system, it is necessary to be familiar with the C++ IPUS platform on which the testbed is implemented. The next section outlines the major features of the C++ IPUS platform.

3.4 C++ IPUS Platform

The C++ IPUS platform provides a convenient facility for developing IPUS applications. In particular, it can be used to construct blackboard data bases, associated knowledge sources, and RESUN control mechanisms (problem solving models, hypotheses, control plans, etc.). The reader is referred to Appendix C for a more detailed description of the platform. In this section, we describe salient features of the C++ platform and how they facilitate the development of application systems.

Selection of the C++ environment for the platform was based on four major factors. First, in contrast to the LISP environment of the original IPUS platform, C++ generally enables very efficient implementations of the numerically intensive computations found in radar applications. Secondly, the C++ platform serves to illustrate that the IPUS architecture specifications have evolved to a level that permits rapid implementation in new software environments. Thirdly, the C++ language brings enables the use of the powerful object-oriented programming paradigm for the design and construction of organized and extensible software systems. And lastly, C++ promises an extremely high level of cross-platform portability due to the ongoing standardization effort coordinated by the American National Standards Institute.

From the viewpoint of the designer of an IPUS application (such as the clutter analysis testbed) the C++ IPUS platform provides a library of base classes of objects from which concrete classes may be derived. The objects are implemented as C++ classes and take advantage of the language features of inheritance, polymorphism, encapsulation, and information hiding. In our platform, base classes are provided for blackboard, hypothesis, problem solving model (PSM), source of uncertainty (SOU), plan, subgoal, and refocus unit objects. These classes embody high-level abstractions of the objects that they represent. To create an instance of an abstract base class, a new *concrete* class must be derived from the base class and definitions provided for its required slots and methods. The process of deriving concrete classes is greatly assisted in the platform by the provision of the *IPUScript* scripting language, which is based on the macro facility of the C++ pre-processor.

A real test of the C++ platform lies in its use for the successful implementation of a radar application. Toward this end, we have implemented the clutter analysis testbed, which utilizes various algorithms and associated radar knowledge developed by our collaborators at Syracuse University. One of the algorithms [6] was only available in the form of a FORTRAN program. Our design of the C++ platform enabled the use of this FORTRAN program in the testbed. In addition to developing the CAT system on the IPUS platform, we implemented an X-Windows based user interface for displaying various intermediate results and final answers produced by the testbed. The CAT system is described in the next section which is then followed by a section on the user interface.

3.5 Clutter Analysis Testbed

The clutter analysis testbed (CAT) serves to demonstrate how the C++ IPUS platform can be used to build applications involving complex signal understanding tasks. We use the term "testbed" because the development of an application on an IPUS platform can proceed *incrementally.* That is, the application developer can experiment with different knowledge sources and control plans to study the effects on the overall problem solving. Furthermore, at any given stage of development an IPUS testbed produces hypotheses and answers which have their respective SOUs attached to them. The developer has the option to add further knowledge to the system in order to deal with those SOUs. With this in mind, we emphasize that while the CAT system can current perform certain interesting clutter analysis tasks, its knowledge base may be conveniently expanded to improve its performance and to extend the range of tasks it can handle.[3] In order to illustrate the current status of the system, we will discuss the CAT blackboard in Section 5.1, the CAT knowledge base in Section 5.2, and the CAT control strategies in Section 5.3.

3.5.1 Blackboard Organization

Data generated during the operation of the CAT system is stored on a blackboard with four information levels. Figure 3.2 shows a pictorial illustration of the blackboard organization utilized in the CAT system. The *return* level on the blackboard is used to record the numerical value of the return obtained by the radar's front-end electronics for each cell in the range-azimuth plane in the form of a two dimensional array. At the next level, a *map* hypothesis is formed. This type of hypothesis is used to describe various properties of each cell's return with respect to the returns in surrounding cells. At the *region* level, hypotheses are formed to describe groupings of contiguous cells with similar *map* level properties. The *patch* level on the blackboard is used to record interpretations of the data at the region level. For example, clutter subregions may be associated with specific types of backgrounds (forest, city skyline, mountain range, etc.).

Each hypothesis on the blackboard is represented in terms of a number of *slots* for recording its attributes. A hypothesis includes *control* slots and *property* slots.

[3]The current capabilities of the CAT system have been illustrated in an on-line demonstration at Rome Laboratories.

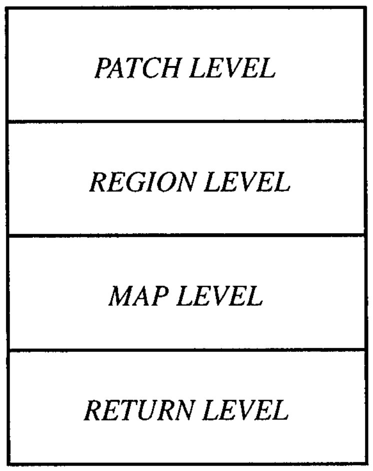


Figure 3.2: The blackboard organization in the Clutter Analysis Testbed.

Control slots are common to hypotheses at all blackboard levels. They are, for example, used to record the name of the KS responsible for posting a hypothesis on the blackboard. The property slots are generally level-dependent. Their purpose is to record application-specific properties of the corresponding hypothesis. In the following discussion of hypotheses at different levels of the CAT blackboard we focus our attention on property slots.

The structure of the hypothesis at the return level is shown in Figure 3.3. As illustrated in the figure it contains a two dimensional array of the size of the range-azimuth plane. Each index of the `data` array is used to store the value of the return obtained by the radar's front-end electronics for the corresponding cell.

```
<RETURN>
  data[rangeIndex][azimuthIndex]  :  numerical value
```

Figure 3.3: The structure for return hypotheses.

The second level contains map hypotheses which are used to store information about the category and statistical characterization of the returns. Figure 3.4 illustrates the structure of a map hypothesis which contains a two dimensional array, named `type`, of the size of the range-azimuth plane. Each index of the array is used to record the category of the return for the corresponding cell. The possible values for this slot are *clutter* and *clear*. The clutter category indicates that the return is primarily due to a background object. The clear category indicates that the return contains primarily electronic noise and/or target components. The value of a return due to a target may be comparable to the value of a return due to a background object. However, returns due to targets tend to be confined in a few isolated cells in contrast to returns due to clutter which extend over several contiguous cells. This fact is used as a distinguishing factor between returns due to targets and returns due to clutter. The uStatistic and the vStatistic slots are used to store information obtained through a statistical analysis for a group of returns that are in the vicinity of the specified cell and that have the same category (clutter or clear) as the category of the return in the specified cell. The information about the group of returns used in the analysis is recorded in the refCells slot. The dist slot includes information about the probability density functions that characterize the returns in this group of returns. All slots are in the form of a two dimensional array as explained for the type slot earlier.

```
<MAP>
  type[rangeIndex][azimuthIndex] :  clutter or clear
  uStatistic[rangeIndex][azimuthIndex] :  0.00023
  vStatistic[rangeIndex][azimuthIndex] :  0.33678
  dist[rangeIndex][azimuthIndex] :  ((pdf1,pdf1-parameters),
                                     (pdf2,pdf2-parameters),...)
  refCells[rangeIndex][azimutIndex] :(Index(32,44), Index(32,45),...)
```

Figure 3.4: The structure for a map hypothesis .

The region hypotheses are placed at the third level of the blackboard hierarchy. These hypotheses describe clutter regions formed in the range-azimuth plane. A clutter region may include subregions within it. A subregion is defined by the property that it includes returns with homogeneous statistical characteristics. Figure 3.5 illustrates the structure used for region hypotheses. The set of cells that are grouped to form a clutter region are recorded in two ways. The RAPlane slot is in the form of a two dimensional array, whereas the cellList slot contains the list of cell indices included in the region. Statistical information obtained for the returns within the entire clutter region are recorded in the regUstat and the regVstat slots. The regDist slot includes information about the probability density functions that characterize the returns within the region. The tileList slot is used to specify the statistical characterizations obtained for a set of partitioned portions within the region. We consider only those partitioned portions which are completely within the region and refer to each of them as a *tile*. Each is represented by an object describing the locations and the statistical characteristics of the tile. The subRegionList slot specifies those portions of the clutter region where the returns have homogeneous statistical characteristics. The structure used for a subregion is essentially the same as the structure of a region hypothesis. The boundaryList and boundaryCells slots specify boundaries between different subregions in terms of boundary tiles and cells respectively.

```
<REGION:1>
  cellList:  (Index(20,31), Index(20,32), ...)
  RAPlane[rangeIndex][azimuthIndex] :  1
  regUStat:  0.00067
  regVStat:  0.32288
  regDist:  ((pdf1,pdf1-parameters),
             (pdf2,pdf2-parameters),...)
  tileList:  (Index(2,3), Index(2,4), ...)
  subRegionList:  (subregion1, subregion2, ...)
  boundaryList:  (boundary1, boundary2, ...)
  boundaryCells:  (Index(35,47), Index(35,48), ...)
```

Figure 3.5: The structure for a region hypothesis.

At the highest level of the blackboard are the patch hypotheses. Figure 3.6 illustrates the representation used for such hypotheses. Patches may be viewed as descriptions of various subregions in region hypotheses in terms of objects in the environment. The name slot specifies the type of object, such as a mountain, a forest, etc., that gives rise to the clutter patch. The patchDist slot in the representation indicates the probability density functions that characterize the returns within the patch. The cellList slot is used to record the boundary of a patch in terms of a list of cells as represented by (range,azimuth) pairs.

```
<PATCH:1>
  patchDist:  ((pdf1,pdf1-parameters,distance1),
               (pdf2,pdf2-parameters,distance2),...)
  cellList:(Index(35,47), Index(35,48), ...)
  name:  ocean, mountain, forest, etc.
```

Figure 3.6: The structure for a patch hypothesis.

Hypotheses such as those at the patch level can be operated upon by knowledge sources (KSs). The contents of these KSs embody the knowledge in the CAT system.

3.5.2 Knowledge Base

The current state of the CAT knowledge base was dictated by our desire to have the system perform two important clutter analysis tasks. These tasks and their associated knowledge were largely developed by the researchers at Syracuse University [5]. The development of the CAT system required incorporation of that knowledge into the appropriate IPUS components. A description of that knowledge and its relationship to various IPUS components is best presented in two parts, each corresponding to a different clutter analysis task.

Task 1

An important clutter analysis task is to separate clutter regions from clear (background noise only) regions. Typically, clutter regions have greater average power than clear regions. However, since the returns in each type of region are probabilistically distributed, some of the returns deviate significantly from the corresponding mean. For example, within a clutter region one often finds isolated returns whose values are well below the clutter mean and comparable to many of the returns in the clear regions. Conversely, within the clear region there can be many isolated values which are well above the noise mean and comparable to many of the returns in the clutter region. To address this problem the researchers at Syracuse University have developed an iterative algorithm [5]. We have broken down the algorithm in terms of the IPUS processing loop of initial processing, discrepancy detection, discrepancy diagnosis and reprocessing. The problem solution in the testbed consists of five major phases:

- *Initial Processing:* This phase requires sorting of the returns by their magnitudes. The returns are then labeled as clutter or clear based on an expected percentage of noise returns. For example, if 10 percent of the returns are expected to be noise only, then after sorting, the bottom 10 percent of the returns are declared to be clear. The top 90 percent are classified as clutter.

- *Discrepancy Detection:* The objective of this phase is to to determine if there are any discrepant noise or clutter cells. A cell is declared discrepant if a sufficient number of its neighbors belong to the opposite category.

- *Discrepancy Diagnosis:* The discrepancy diagnosis is performed to determine if the discrepant cells have been simply mislabled or if they may have arisen due to an incorrect assumption about the percentage of noise cells in the entire range-azimuth plane.

- *Reprocessing Planning:* If the diagnosis is that the discrepant cells are simply a result of mislabeling, relabeling of the cells is planned. Otherwise, a knowledge base of rules is used to decide how to adjust the assumed percentage of noise cells. The rules are also used to produce recommendations regarding the number of neighboring cells to be examined for identifying each discrepant clutter or noise cell.

- *Reprocessing:* The reprocessing plan is carried out, to be followed once again by discrepancy detection and, if need be, by discrepancy diagnosis and further reprocessing planning.

The task of separating clutter and clear regions is a good but simple illustration of incorporating problem solving knowledge in the IPUS paradigm. The algorithm described above has been successfully implemented in the CAT system. The separation of clutter and clear regions is typically achieved in four to six iterations. In Section 5.3, we describe the control strategy built into the testbed for this task. In the following section we consider the knowledge base included in the CAT system for a significantly more complex clutter analysis task.

Task 2

Another component of the knowledge base in the CAT system concerns the task of separating homogeneous subregions within hypothesized clutter regions. In particular, we focus upon the case where the boundary between two homogeneous regions is at an approximately constant range over a wide sector of azimuth angles. We will refer to such a boundary as a constant-range boundary. The difficulty here is that while knowledge of the boundary is necessary for verifying the homogeneity for the regions on either side, knowledge about the homogeneity characteristics is necessary for determining the boundary. This suggests the desirability of an approach in which initial processing for an approximate localization of the boundary is followed by knowledge-based reprocessing for refining the boundary. Next we illustrate how existing signal processing techniques can be utilized for carrying out this type of analysis. Section 5.3 details the corresponding control strategy implemented in the CAT system.

Relevant Signal Processing Techniques

Our approach to the problem of determining a constant-range boundary within a clutter region consists of five major phases.

- *Initial processing:* The entire clutter region under consideration is partitioned into rectangular subregions referred to as *tiles.* Each of these tiles is processed in order to obtain a corresponding statistical characterization. This processing is carried out under the assumption that each tile is homogeneous.

- *Discrepancy detection:* The objective of this phase is to determine if the homogeneity assumption is violated for any of the tiles. For this purpose, it is necessary that the tile size should have been selected to be much smaller than any of the homogeneous subregions of the clutter region under analysis. Discrepancy detection involves the identification of each tile whose statistical characterization is sufficiently different from the statistical characterization of its neighboring tiles.

- *Discrepancy diagnosis:* The main purpose of this phase is to determine whether the existence of any discrepant tiles can potentially be explained on the basis of a constant-range boundary. For this purpose we utilize the fact that a constant-range boundary would give rise to a group of discrepant tiles along the boundary. For this group, the neighboring tiles at higher ranges would have the property that their statistical characterizations are similar to each other. Furthermore, the neighboring tiles at lower ranges would also have statistical characterizations which are similar to each other. Of course, the statistical characterization common to the higher-range neighbors would be different from the statistical characterization common to the lower-range neighbors. If such a pattern is found to exist, we hypothesize that each of the discrepant tiles contains a constant-range boundary within it.

- *Reprocessing Planning:* This phase determines the kind of reprocessing that can be performed in order to refine the boundary. For example the reprocessing may be planned on the basis of empirical knowledge about the characteristics of the distributions involved.

- *Reprocessing:* The goal of this phase is to refine the various boundary hypotheses formed during discrepancy diagnosis based on the strategy developed in reprocessing planning. The data in each discrepant tile is combined with the data in its higher-range neighbor and the data in its lower-range neighbor. A search is then performed to determine how the hypothesized boundary within that data ought to be adjusted in order to ensure that the data on each side of the boundary is homogeneous.

In order to discuss each of the five phases in greater detail, we consider a specific example involving a clutter region with a constant-range boundary. For this example, we synthesized a clutter region in the range-azimuth plane, which gets partitioned into 24 tiles of 100 cells each. As illustrated in Figure 3.7, an approximately horizontal boundary divides this region into two subregions. The returns in the higher-range subregion were generated as independent samples of a random variable with a probability density function (PDF) of the Rayleigh class:

$$f_R(x) = \frac{x}{\beta^2} e^{-(\frac{x^2}{2\beta^2})} \quad x \geq 0 \tag{3.1}$$

where the scale parameter (β) was selected to have a value of 2.23. This results in a value of 3.16 for the second order moment. The returns in the lower-range subregion were generated as independent samples of a random variable with a PDF of the Lognormal class:

$$f_L(x) = \frac{\gamma}{\sqrt{2\pi}x} e^{-[\gamma log(x/\beta)]^2/2} \qquad x \geq 0 \tag{3.2}$$

where the shape parameter (γ) was selected to have a value of 1.0, and the scale parameter (β) was selected to have a value of 3.15. This results in a value of 3.16 for the second order moment. Since the two subregions have identical second-order moments, it is not possible to detect the boundary between them through some energy thresholding method.

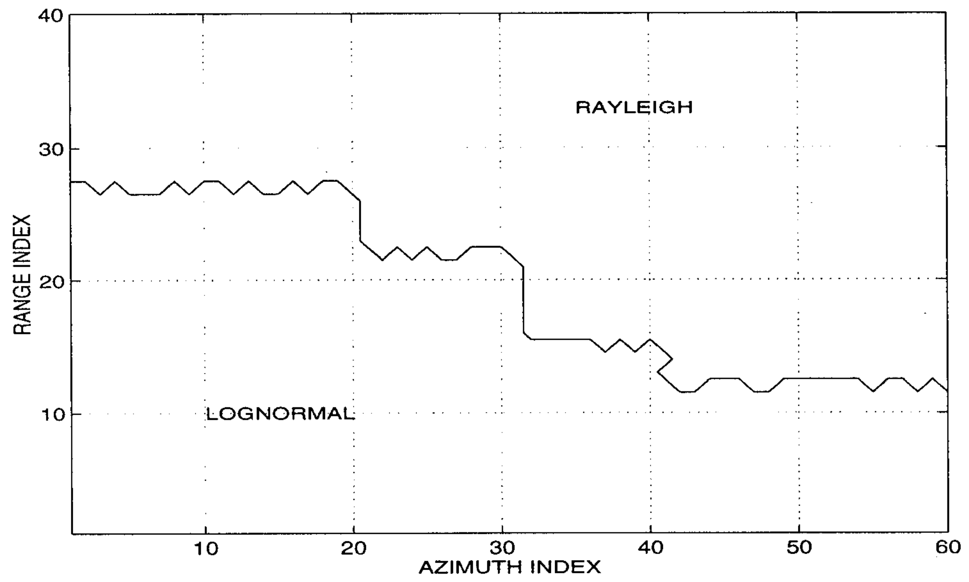


Figure 3.7: The test scenario. The data was generated as samples of a Rayleigh distribution and Lognormal distribution with a shape parameter of 1.0.

Initial Processing: The clutter region under consideration is partitioned into tiles of size 10×10, as illustrated in Figure 3.7. A statistical signal processing algorithm [6] is applied to the returns in each tile. This algorithm (referred to as Ozturk's algorithm) produces outputs which can be used for the statistical characterization of the input tile. To describe the nature of these outputs in greater detail, it is necessary for us to establish some notation and associated terminology.

Let the 100 returns in the input tile be represented by the sequence $x[n]$, where $0 \leq n < 100$. Assume that each return, $x[n_0]$, is a sample of an independent random variable $X[n_0]$. Furthermore, these random variables are assumed to be identically distributed. The corresponding PDF is denoted the function $f_X(x; \gamma; \alpha, \beta)$, where x is the independent variable, γ is a shape parameter, α is a location parameter, and

β is a scale parameter[4]. The ultimate objective of Ozturk's algorithm is to identify a PDF (including the values of the shape, location and scale parameters) which can be considered a good approximation to the PDF of $X[n]$. Toward this end, the sequence $x[n]$ is first transformed into a sequence $y[n]$ as follows:

$$y[n] = \frac{x[n] - \overline{x}}{S} \quad ; \quad n = 0, 1, 2, ..., 99 \tag{3.3}$$

where

$$\overline{x} = \frac{1}{100} \sum_{n=0}^{99} x[n] \tag{3.4}$$

is the sample mean and

$$S = \left[\frac{1}{100} \sum_{n=0}^{99} (x[n] - \overline{x})^2 \right]^{1/2} \tag{3.5}$$

is the sample standard deviation. The elements of the sequence $y[n]$ can be considered the respective samples of independent random variables $Y[n]$. Each of these random variables has a PDF which is identical to the PDF associated with $X[n]$ except for having a zero mean and unity variance. The ordering of the elements of the sequence $y[n]$ is then changed to obtain a sequence $z[n]$. This sequence has its elements in order from the smallest value (at $n = 0$) to the largest value (at $n = 99$). The nth element of $z[n]$ can be considered as a sample of the *order* statistic of rank n corresponding to the 100 random variables represented by $Y[n]$. Two quantities, U and V, are then calculated from $z[n]$ as follows:

$$U = \frac{1}{100} \sum_{i=0}^{99} cos(\theta_i)|z[n]| \tag{3.6}$$

and

$$V = \frac{1}{100} \sum_{i=0}^{99} sin(\theta_i)|z[n]| \tag{3.7}$$

where

$$\theta_i = \pi\Phi(m_i) \tag{3.8}$$

$\Phi(x)$ is the distribution function of the standard Gaussian distribution, and m_i for $i = 0, 1, 2, ..99$ denote the expected values (obtained through Monte Carlo trials) of the standard Gaussian *order* statistics. The values U and V specify a point

[4]It is assumed that the r.v. X belongs to a family of parameterized PDF's. The location and scale parameters represent the relationship of X to a standard member Y of the family through $X = \beta Y + \alpha$.

$Q = (U, V)$ in a U-V plane. It has been shown [6] that a variety of well-known PDF's (with zero mean and unity variance) tend to occupy different locations in the U-V plane. Furthermore, a PDF family having a shape parameter gives rise to a continuous trajectory in the $U - V$ plane. It follows that the values of the U-statistic and the V-statistic can be used as the basis for hypothesizing a tile's PDF. A particular procedure for this purpose is reported elsewhere [6].

The (U, V) measurements obtained for various tiles in the scenario of Figure 3.7 are illustrated in Figure 3.8. The (U, V) measurements indicated by the symbol "o" correspond to tiles in the Lognormal region, the measurements denoted by the symbol "×" correspond to tiles in the purely Rayleigh distributed region, and the measurements shown by the symbol "*" are associated with the tiles that lie on the boundary between these two regions.

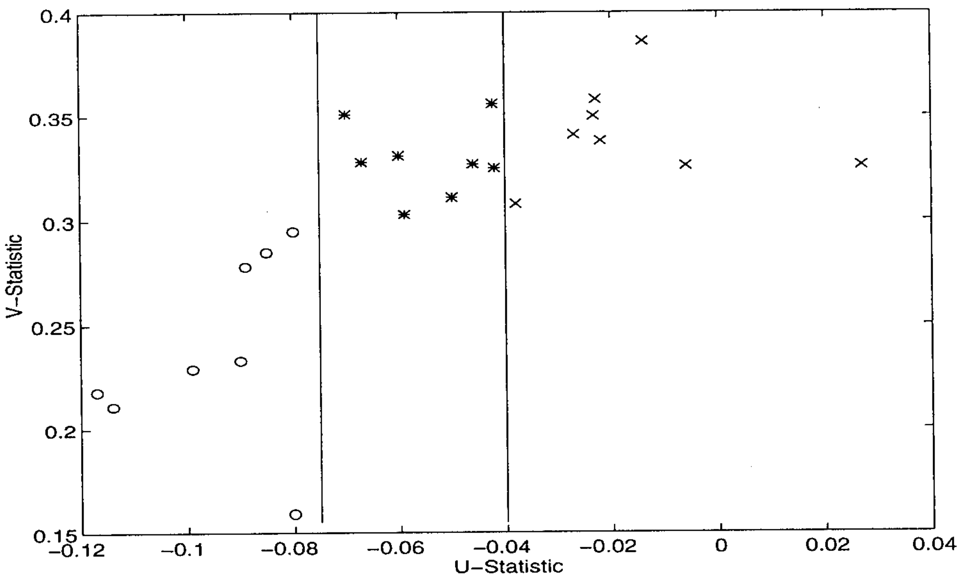


Figure 3.8: The (U, V) measurements obtained for various tiles in the test scenario. The measurements indicated by the symbol "o" correspond to tiles in the Lognormal region, the measurements denoted by the symbol "×" correspond to tiles in the purely Rayleigh distributed region, and the measurements shown by the symbol "*" are associated with the tiles that lie on the boundary between these two regions.

Discrepancy Detection: We perform discrepancy detection in order to find the tiles which violate the homogeneity assumption. This involves the grouping of tiles with (U, V) measurements falling within certain limits. A tile which is not a member of a sufficiently large group is labeled as *discrepant.* In the grouping process for the example case, we check a tile's (U) statistic against the two range of values for the two distributions. The tiles falling within the same range of values are grouped together. If a group does not span at least two tiles in both range and azimuth dimensions, its member tiles are declared to be discrepant. The cells in each of the remaining groups are said to constitute a homogeneous subregion. In our test scenario, this procedure results in two homogeneous subregions, R and L, whose tiles are labeled "+" and "o", respectively, in Figure 3.9. Also, the discrepant tiles are indicated by the symbol "*".

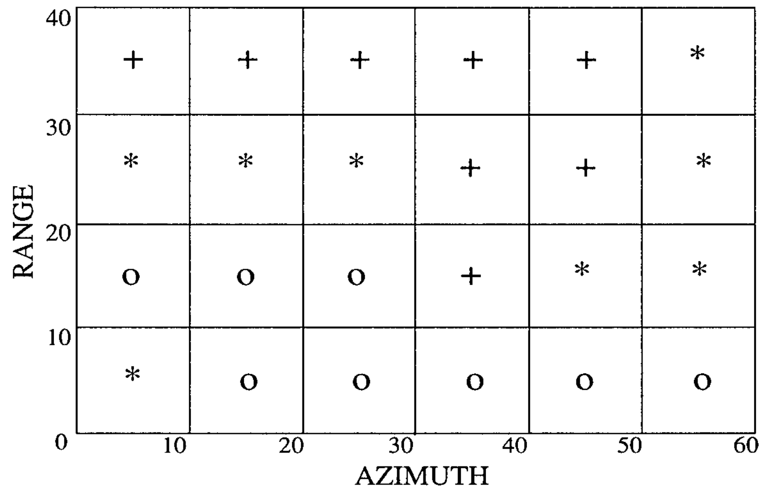


Figure 3.9: The groups of tiles in the range-azimuth plane. The two homogeneous subregions, W and L, are labeled by "+" and "o", respectively. The discrepant tiles are indicated by the symbol "*".

Discrepancy Diagnosis: The main purpose of discrepancy diagnosis is to determine whether the existence of discrepant tiles may be explained by a constant-range boundary within the clutter region. A constant-range boundary divides a clutter region into two homogeneous subregions. The tile-based initial processing of the radar returns gives rise to a group of discrepant tiles which straddle the boundary, as illustrated in Figure 3.9. The horizontal boundary is determined in two phases. First, we perform a procedure to "track" such discrepant tiles along the azimuthal dimension. If t_k is the latest tile to be added to the track, further extension of the track requires the examination of the three adjacent tiles, t_{k+1}, t_{k+2}, and t_{k+3} as illustrated in Figure 3.10. The track is extended only if one of these tiles is discrepant. The formation of such a track is considered sufficient evidence for hypothesizing a constant-range boundary as an explanation for the discrepant tiles. In our test scenario, the isolated discrepant tiles that do not form part of a horizontal boundary are considered to be part of the adjacent subregions. In the second phase, we determine if two horizontal boundaries can be connected by assuming a non-discrepant tile to be discrepant. Such a situation in our test scenario is depicted in Figure 3.11 where the tiles considered to be part of the boundary are marked by diagonal crosses. Note that the two horizontal boundaries obtained in the first phase are connected by declaring a Rayleigh distributed tile to be discrepant.

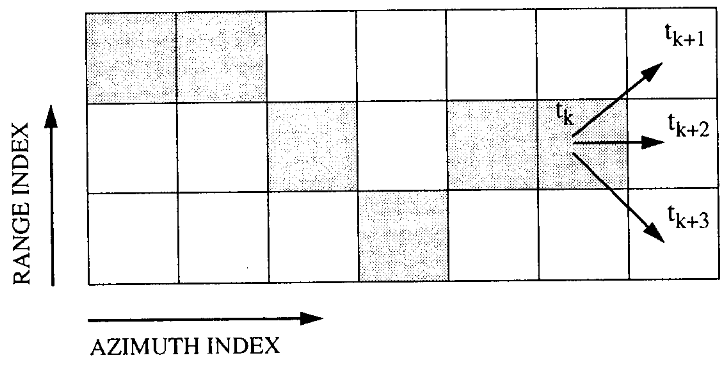


Figure 3.10: The tracking of discrepant tiles. The shaded tiles form a track that has been extended up to a tile t_k. The track is further extended if one of the tiles, t_{k+1}, t_{k+2}, and t_{k+3}, is discrepant.

Reprocessing Planning: This phase determines the kind of reprocessing that can be performed in order to refine the boundary. For example the reprocessing may be planned on the basis of empirical knowledge about the characteristics of the distributions involved. In our test scenario it was assumed that the boundary passes through the middle of each discrepant tile. Therefore the cells falling on the fifth row of a discrepant tile were included in the boundary of the lower homogeneous subregion belonging to the Lognormal class of distribution. The cells in the sixth row were put on the boundary list of the upper homogeneous region of the Rayleigh distribution class. The boundary obtained by processing the discrepant tiles in the manner described is shown in Figure 3.12.

Reprocessing: The objective of signal reprocessing is to reduce the uncertainty associated with the boundary estimates produced by the discrepancy diagnosis procedure. For each boundary tile t_0 one may, for example, want to estimate the location of the horizontal boundary to within a resolution of two cell widths in the range dimension. One approach is to hypothesize that the top 2,4,6, or 8 rows in the

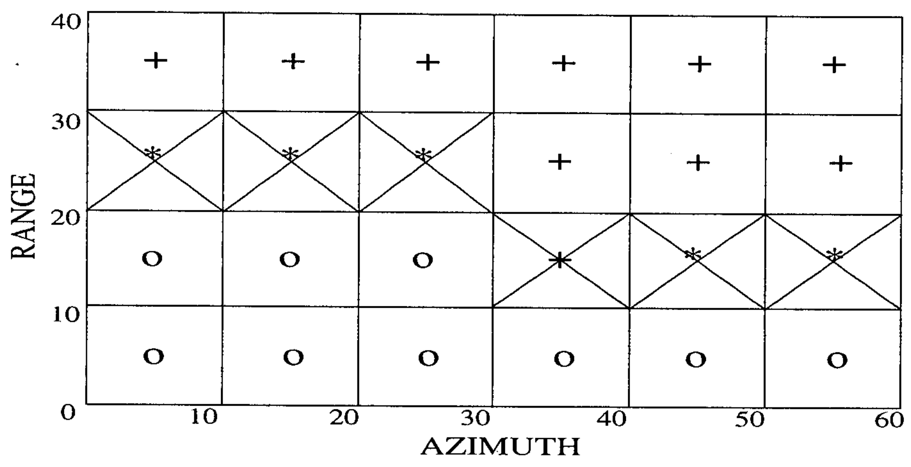


Figure 3.11: Horizontal boundary. The two separate horizontal boundaries are connected by declaring a "+" tile to be discrepant. The tiles included in the boundary are shown with a cross.

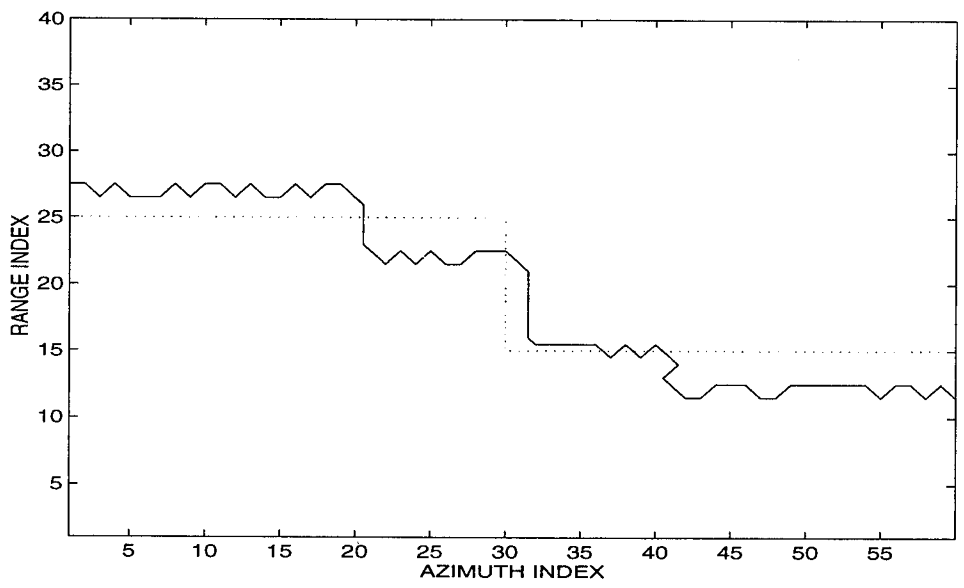


Figure 3.12: The original boundary (solid line) and the boundary obtained after signal processing (dotted line).

discrepant tile belong to the upper homogeneous subregion. Denoting this number of top rows in t_0 by N_i ($i = 1, 2, 3, 4$), a test may be conducted for choosing among the corresponding boundary hypotheses. In this test, two new tiles (denoted by t_{i1} and t_{i2}, respectively) are formed for each of the four possible values for N_i. As

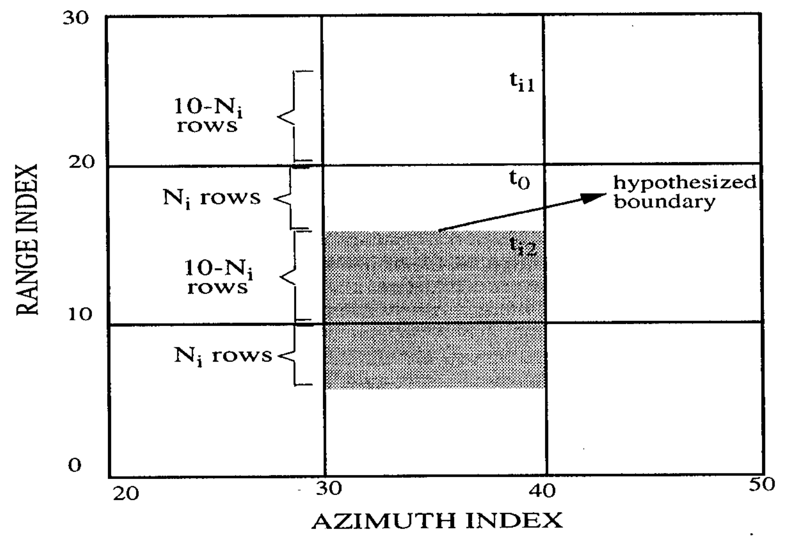


Figure 3.13: The formation of two tiles, t_{i1} and t_{i2}, under the hypothesis that the top N_i rows in a tile, t_0, belong to the subregion located above the tile.

illustrated in Figure 3.13, the tile t_{i1} is a combination of the top N_i rows of t_0 with $(10 - N_i)$ rows of the tile directly above it. The tile (t_i2) is a combination of the bottom $10 - N_i$ rows of tile t_i1 with N_i rows of the tile directly below it. For each hypothesis, N_i, the measurement for tile t_i1 , $(U_{t_i1}^{N_i}, V_{t_i1}^{N_i})$, and the measurement for tile t_i2, $(U_{t_i2}^{N_i}, V_{t_i2}^{N_i})$, are obtained. The distances between these measurements and the measurements for neighboring homogeneous subregions are calculated. In the case of our test scenario, these distances are defined as:

$$d_{t_{i1},R} = \sqrt{(U_{t_{i1}}^{N_i} - U_R)^2 + (V_{t_{i1}}^{N_i} - V_R)^2} \tag{3.9}$$

and

$$d_{t_{i2},L} = \sqrt{(U_{t_{i2}}^{N_i} - U_L)^2 + (V_{t_{i2}}^{N_i} - V_L)^2} \tag{3.10}$$

where (U_R, V_R) and (U_L, V_L) are the measurements associated with the subregion R and the subregion L, respectively. The hypothesis N_i, for which the cumulative distance $d_C = d_{t_{i1},R} + d_{t_{i2},L}$ is minimized, constitutes the most consistent hypothesis.

The procedure results in narrowing the possible location of a horizontal boundary within a given discrepant tile.

Having established the nature of the knowledge base embodied in the various CAT knowledge sources, we shall examine the currently implemented CAT control plans in the next section. As alluded to earlier, the IPUS architecture permits the application designer to incrementally expand and/or modify these control plans as well as the associated knowledge sources.

3.5.3 Control Plans

An IPUS implementation of the CAT tasks introduced in the previous section also requires specification of a control strategy. This control knowledge is encoded in the goal/plan/subgoal structures associated with the RESUN planner of the IPUS platform.

Control Strategy for Task 1

In this section, we describe how the CAT system goes about categorizing the returns. Specific control strategy aimed at performing this task is explained with the help of figures.

The processing in the system is driven by the various sources of uncertainty (SOUs) attached with the hypotheses at a given time. When the return hypothesis is created at the lowest blackboard level, it is tagged with a `NoExplanationSOU`. The system then aims to satisfy the high level subgoal of `HaveHypothesisSOUSolved`. The input to the subgoal is the SOU which needs to be resolved. It is used as the input constraint parameter `inConstraint` to find a matching plan for the subgoal. In this case the plan `SolveNoExplanationReturnHyp` matches the subgoal. The plan in turn posts two subgoals to be met sequentially as illustrated in Figure 3.14. The first subgoal `HaveInitialThreshold` is matched by the primitive plan `GetInitialThreshold` which retrieves the preset threshold value in terms of the expected percentage of noise returns in the range-azimuth plane. The second subgoal `HaveClutterMarked` is matched by the primitive plan `MarkClutter`. This KS performs the function of marking the returns as clutter or clear based on the threshold and fills in the `type` slot of the map hypothesis. The initial assignment of clutter or clear needs to be verified. Therefore a `PartialVerificationSOU` is posted with the map hypothesis whereas, the `NoExplanationSOU` is removed from the return hypothesis.

The presence of `PartialVerificationSOU` with the map hypothesis causes the high level `HaveHypothesisSOUSolved` subgoal to be posted again. Since the input variable is the SOU, the plan `SolvePartialVerificationSOU` matches the subgoal. This plan posts the following subgoals to be met sequentially in a loop until the termination condition is satisfied (Figure 3.15):

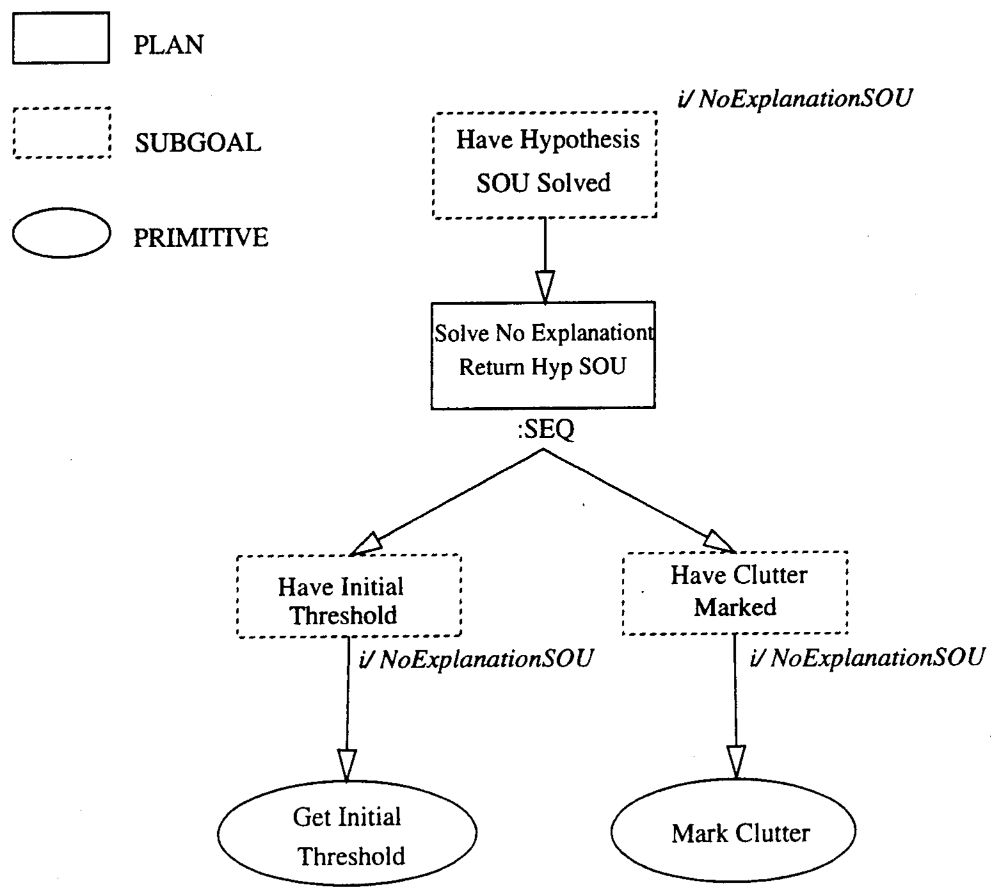


Figure 3.14: A pictorial representation of control plan strategy for pursuing a subgoal, `HaveHypothesisSOUSolved` with input `NoExplanationSOU` attached to a *return* hypothesis.

- `HaveDiscrepancyDetection`.
- `HaveDiscrepancyDiagnosis`.
- `HaveReprocessingPlanning`.
- `HaveReprocessing`.

The first subgoal is matched by the plan `DiscrepancyDetectionMapHyp` which posts the subgoal `HaveCellsRecovered`. This subgoal invokes the KS `RecoverCells` which detects the presence of isolated clutter and clear cells through smoothing and recovers them *ie.* relabels them otherwise.

The `HaveDiscrepancyDiagnosis` subgoal is matched by the `DicrepancyDiagnosisMapHyp` plan which computes the percentage of clear cells in the original quantized and the corrected volumes.

Subgoal `HaveReprocessingPlanning` invokes the `ReprocessingPlanning` KS which essentially performs the assessment. If the percentage of clear cells in the quantized and corrected volumes are consistent, the loop termination condition is set and the `PartialVerificationSOU` of the map hypothesis is replaced by the `NoExplanationSOU`. In the case of inconsistency a new threshold based on the difference in the percentage of clear cells in the two volumes is computed.

Finally the subgoal `HaveReprocessing` is matched by the `MarkClutter` KS which performs the task of marking the returns as clutter or clear based on the new threshold.

Control Strategy for Task 2

In this section, we specify how a system for clutter analysis may go about finding inhomogeneities in a clutter region during any phase of its problem solving activities. As specified in the sample task definition, such inhomogeneities are indicated by the presence of boundaries separating homogeneous subregions. The control strategy we have developed aims at finding such homogeneous subregions and the constant-range boundaries separating them.

The specification of the control strategy we have developed begins with the graph in Figure 3.16. The system aims at satisfying the high level subgoal `HaveHypothesisSOUSolved` in order to extend the region hypothesis tagged with the `PartialSupportSOU`. For this purpose, a corresponding plan `SolvePartialSupportRegionHypSOU` is retrieved. This plan posts the subgoal `HaveTileCharacteristics` which invokes the KS `FindTileCharacteristics`. The input to this plan is the SOU being resolved which indicates the specific region hypothesis for which tile characteristics are to be obtained. The KS divides the region into tiles of size 10×10 and runs the FORTRAN Ozturk's algorithm to obtain the statistical characteristics of each tile. Only those tiles are processed that fall completely inside the region. In order to verify the homogeneity assumption

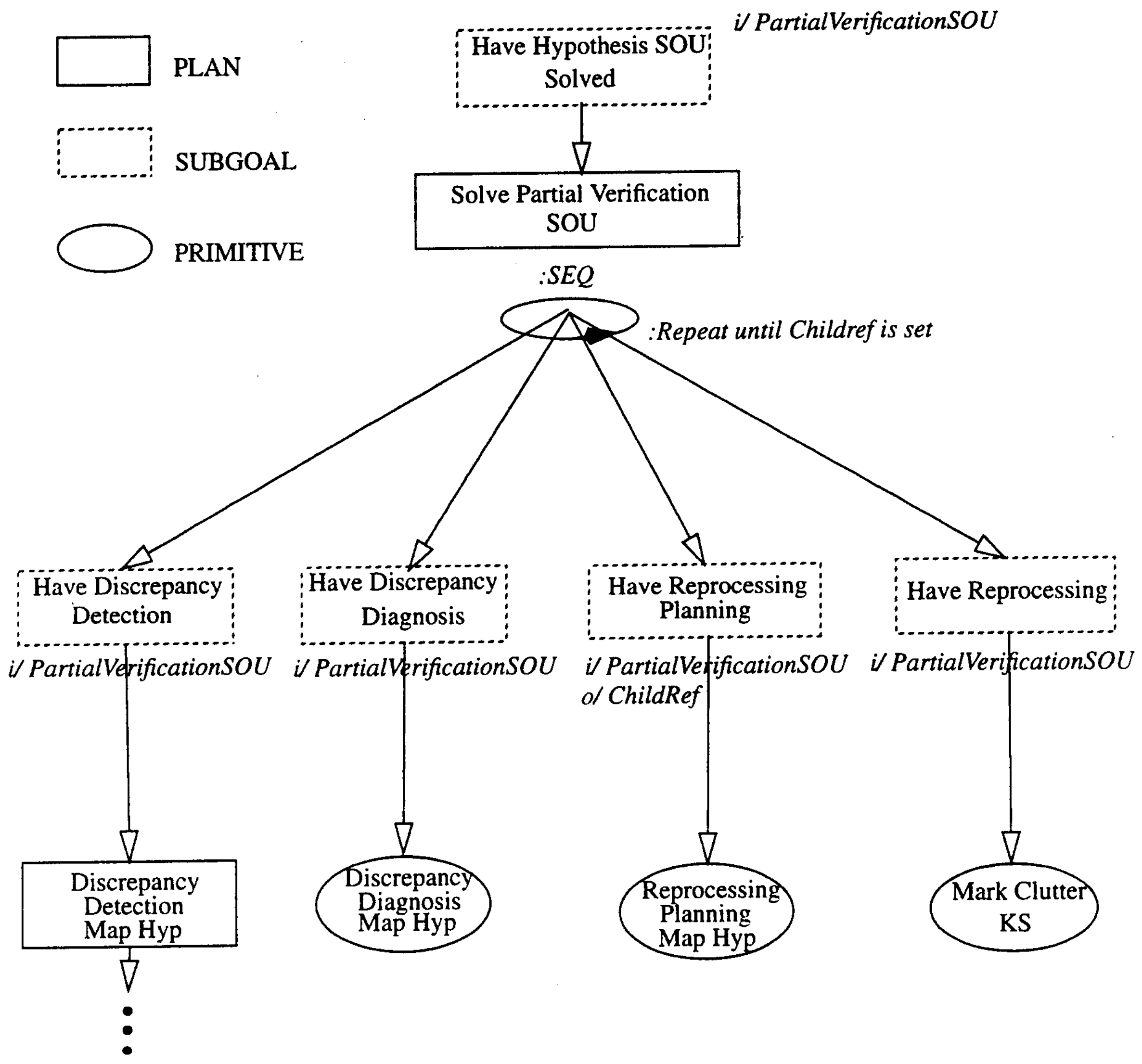

PLAN
SUBGOAL
PRIMITIVE
i/ PartialVerificationSOU
Have Hypothesis SOU Solved
Solve Partial Verification SOU
:SEQ
:Repeat until Childref is set
Have Discrepancy Detection
i/ PartialVerificationSOU
Have Discrepancy Diagnosis
i/ PartialVerificationSOU
Have Reprocessing Planning
i/ PartialVerificationSOU
o/ ChildRef
Have Reprocessing
i/ PartialVerificationSOU
Discrepancy Detection Map Hyp
Discrepancy Diagnosis Map Hyp
Reprocessing Planning Map Hyp
Mark Clutter KS

Figure 3.15: A pictorial representation of the plan, `HaveHypothesisSOUSolved` with input `PartialVerificationSOU` attached to a *map* hypothesis.

about the region the PartialVerificationSOU is attached to the region and the PartialSupportSOU is removed.

When the higher level subgoal of HaveHypothesisSOU solved is matched by the SolvePartialVerificationSOU plan, it posts four subgoals to be pursued sequentially as described earlier in the control strategy for sample task 1. Since the input variable the PartialVerificationSOU is attached to a region hypothesis, different plans match these subgoals this time. These plans are relevant to the discrepancy detection, diagnosis and reprocessing at the region level (Figure 3.17). The first subgoal HaveDiscrepancyDetectionis aimed at performing discrepancy detection and is matched by the DiscrepancyDetectionRegionHyp primitive plan. This KS finds contiguous tiles with characteristics that fall within prespecified ranges and groups them together accordingly into homogeneous subregions. The tiles which do not fall within a group are declared discrepant. If there are any such discrepant tiles, we pursue the second subgoal for performing discrepancy diagnosis. This subgoal is satisfied by activating the discrepancy diagnosis KS DiscrepancyDiagnosisRegionHyp, whereby the presence of constant-range boundaries is determined. The subgoal for performing reprocessing planning is activated only when the discrepant tiles are explained by the existence of constant-range boundaries. When activated it is matched by the KS ReprocessingPlanningRegionHyp. In the current implementation this KS creates the boundary between subregions based on the assumption that it passes through the middle of discrepant tiles. The PartialVerificationSOU is removed from the region hypothesis and the NoExplanationSOU is attached in order to drive the processing further to the patch level.

3.6 User Interface

The CAT system has been developed with a comprehensive user interface. The interface uses *X-Windows* for displaying information about the operation of the testbed. The display consists of two text windows and one graphics window with several click-on buttons for controlling the display. The text windows display information about the status of the PSM and the executed KSs at any instant during the processing. The graphics window is used to graphically display various hypotheses on the blackboard. The display environment also serves as a good debugging tool for the development of the testbed. The feedback provided by the display helps the designer in uncovering problems and errors. Apart from the display environment built on the CAT system, the C++ IPUS platform provides the facility of a "trace" output. The trace gives detailed information about the processing in the testbed. A description of this facility can be found in [3].

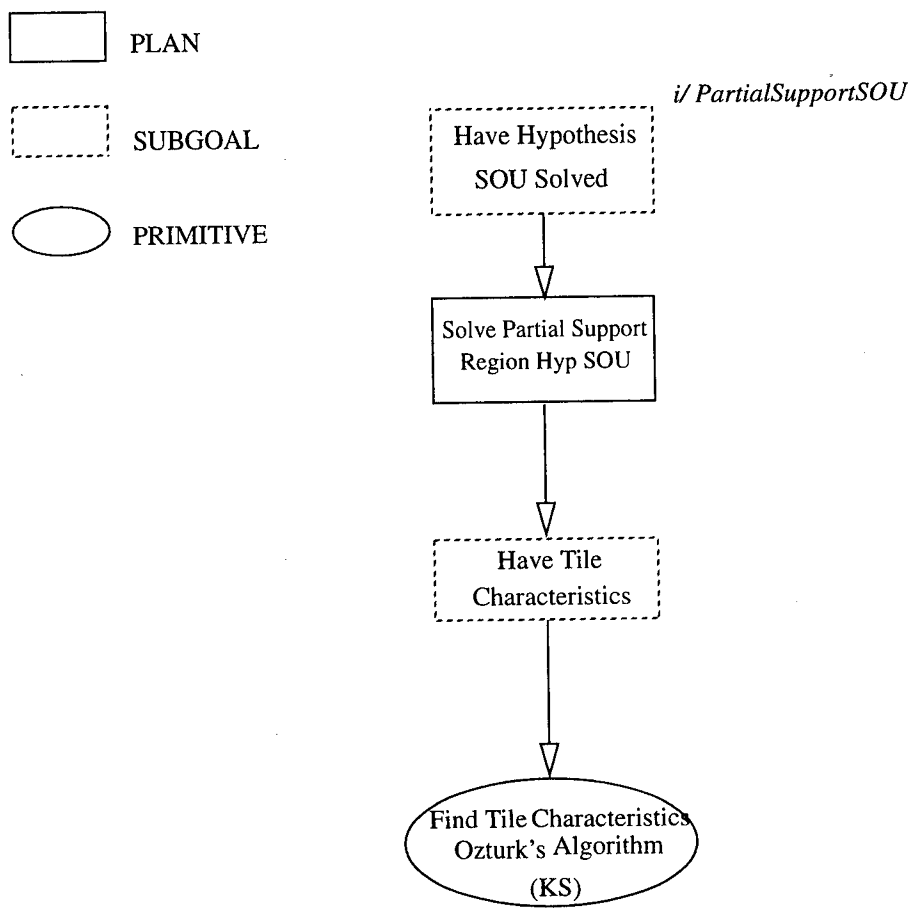

PLAN
SUBGOAL
PRIMITIVE
i/ PartialSupportSOU
Have Hypothesis SOU Solved
Solve Partial Support Region Hyp SOU
Have Tile Characteristics
Find Tile Characteristics Ozturk's Algorithm (KS)

Figure 3.16: A pictorial representation of the plan, `HaveHypothesisSOUSolved` with input `PartialSupportSOU` attached to a *region* hypothesis.

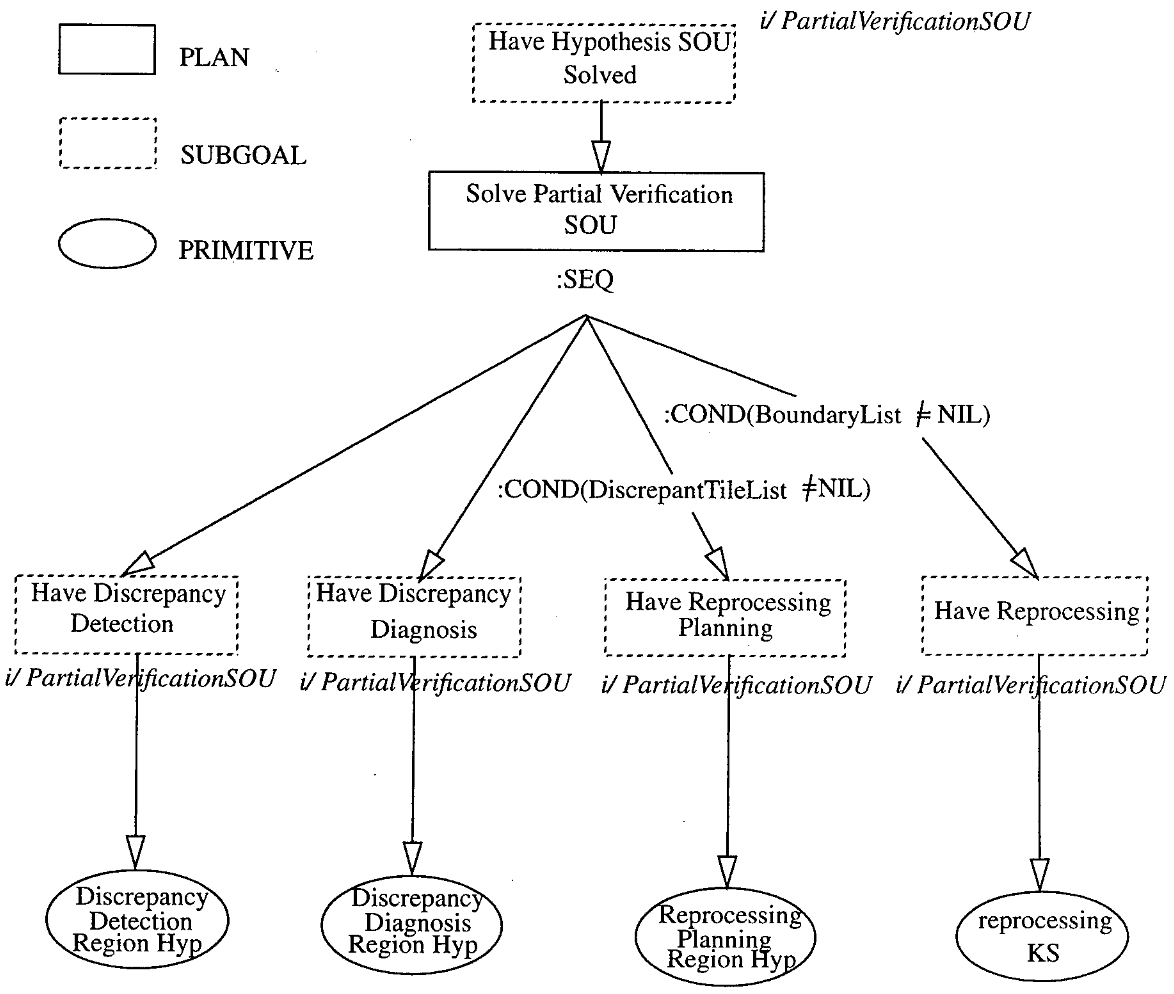

PLAN
SUBGOAL
PRIMITIVE
Have Hypothesis SOU Solved
i/ PartialVerificationSOU
Solve Partial Verification SOU
:SEQ
:COND(BoundaryList ≠ NIL)
:COND(DiscrepantTileList ≠NIL)
Have Discrepancy Detection
Have Discrepancy Diagnosis
Have Reprocessing Planning
Have Reprocessing
i/ PartialVerificationSOU
i/ PartialVerificationSOU
i/ PartialVerificationSOU
i/ PartialVerificationSOU
Discrepancy Detection Region Hyp
Discrepancy Diagnosis Region Hyp
Reprocessing Planning Region Hyp
reprocessing KS

Figure 3.17: A pictorial representation of the plan, `HaveHypothesisSOUSolved` with input `PartialVerificationSOU` attached to a *region* hypothesis.

3.7 Summary and Conclusions

Our radar oriented IPUS research has resulted in the development of three major software systems. The C++ IPUS platform provides the generic capabilities for implementing application systems. The clutter analysis testbed (CAT) is a concrete illustration of an application system built on top of the C++ IPUS platform. Finally, the X-windows based user interface permits both testbed developers and testbed users to conveniently inspect the results produced by the CAT system. The successful development of the platform, the testbed, and the user interface is indicative of the on-going maturation of IPUS technology. We have succeeded in transfering IPUS technology from one software environment to another and from an acoustic context to a radar context. This sets the stage for more detailed investigations for evaluating the impact IPUS technology can have on particular radar applications. Some of the directions for such investigations include:

- Further expansion of the knowledge base in the CAT system and subsequent evaluation of its performance for real radar data.
- Incorporation of auxilary information (such as terrain maps) in the CAT system to evaluate more of the top-down processing features of the testbed.
- Utilization of the *refocus units* capability in the C++ platform to provide a greater reactive component to the planner in the CAT system.
- Development and performance evaluation of testbeds for other radar applications.
- Comparison of the performance of IPUS systems against other systems for the same applications.
- Revisions and updates for the C++ IPUS platform to take advantage of the latest research.
- Further development of the CAT user interface to facilitate more detailed examination of the system's problem solving.
- Development of software tools to further aid the development of applications on the C++ IPUS platform.

The directions listed above should yield quantitative evaluations of the impact that IPUS technology can have on radar applications involving sophisticated interaction between signal processing and signal interpretation. However, as already demonstrated by our development of the CAT system, IPUS technology also enables complex signal processing strategies to be implemented methodically and conveniently.

Bibliography

[1] S. H. Nawab and V. Lesser, "Integrated Processing and Understanding of Signals," in *Symbolic and Knowledge-Based Signal Processing*, eds. A. V. Oppenheim and S. H. Nawab, pp. 251-285, Prentice-Hall: Englewood Cliffs, N.J., 1991.

[2] N. Carver and V. Lesser, "A Planner for the Control of Problem-Solving Systems," *IEEE Trans. on Systems, Man, and Cybernetics,* v. 23, n. 6, pp. 1519-1536, 1993.

[3] J. M. Winograd, "C++ IPUS Platform User's Manual," in this publication.

[4] M.I. Skolnik, *Introduction to Radar Systems,* McGraw-Hill, New York, N.Y., 1980.

[5] M.A. Slamani, "A New Approach to Radar Detection Based on the Partitioning and Statistical Characterization of a Radar Surveillance Volume", PhD thesis, Dept. of Electrical and Computer Engineering, Syracuse University, 1994.

[6] A. Ozturk, E.J. Dudewicz, "A New Statistical Goodness of Fit Test Based on Graphical Representation", TR 152, Dept. of Mathematics, Syracuse University, 1990.

Chapter 4

SIRP Modeling of Non-Gaussian Interference

4.1 Introduction

Communication systems must typically operate in non-Gaussian noise environments which may consist of interference from natural sources and both intentional and unintentional man-made sources. The Gaussian receiver is often reported to exhibit very robust performance in many applications involving non-Gaussian interference. However, this conclusion is usually drawn by comparing the performance of the Gaussian receiver in non-Gaussian noise to its optimal performance in Gaussian noise. The point often overlooked is that the use of a Gaussian receiver in such a non-Gaussian environment can cause a significant performance degradation compared to the optimal non-Gaussian receiver. In such situations it would actually be preferable to operate the system in a non-Gaussian noise environment, *if* the appropriate optimal processor can be selected and applied to the received signal.

The above distinction between selection and application of the optimal processor emphasizes a major complication of optimal non-Gaussian processing. Specification of an optimal Gaussian processor has implicitly identified the joint probability density function (pdf) of the noise samples to be Gaussian. However, when the noise is known to be non-Gaussian, the receiver must first select a pdf which suitably approximates the joint pdf of the interference, before the optimum processing can be determined and applied to the received data. This problem is difficult enough for univariate density functions, due to the infinite number of non-Gaussian pdfs from which to choose. It becomes even more difficult when multivariate pdfs of the samples must be considered.

The traditional development of optimum non-Gaussian processing for communication signals has assumed statistically independent samples of the noise process. This assumption is often not satisfied, but it allows formulation of a usable mathematical solution for the optimum processor and is usually motivated by lack of sufficient mathematical models to completely describe the multivariate, non-Gaussian pdfs of correlated interference samples.

Recent research at Syracuse University has successfully applied the theory of spherically invariant random processes (SIRPs) to correlated, non-Gaussian clutter modeling for radar applications [1, 2]. SIRP models have proven to be very useful because they have many of the same properties as Gaussian processes and their structure is very suited to radar clutter phenomena. Another important advantage of the SIRP clutter model is its reduction of the joint pdf approximation problem to a univariate pdf approximation problem, even for correlated noise samples. This allows straightforward use of the Ozturk algorithm [3, 4], which was developed at Syracuse University and is a very powerful technique for selecting suitable non-Gaussian density functions to approximate the pdf of the received interference.

This report focuses primarily on an investigation into the applicability of SIRPs to modeling non-Gaussian interference in communication systems and the derivation of the corresponding optimum receivers. It also briefly discusses the potential advantages of using the Ozturk algorithm to perform pdf approximation for the conventional model of independent noise samples, as well as for the SIRP models. Unfortunately, due to certain characteristics of the decision process in communication applications and to certain characteristics of man-made interference, it is concluded that SIRPs are not appropriate models to use, except under very restrictive circumstances.

The report is organized in the following way. Section 4.2 is a brief review of key concepts in the theory of spherically invariant random processes and spherically invariant random vectors. Section 4.3 presents a short overview of locally optimum detectors and their application to weak signal detection problems, particularly in the presence of spherically invariant interference. Section 4.4 describes the nature of the non-Gaussian interference, with emphasis on intentional jamming, and discusses possible problems of using SIRP models. Section 4.5 investigates the structure of the minimum probability of error communication receiver in spherically invariant interference for the M-ary decision problem. Results for several cases of unknown amplitude and phase parameters are presented. Section 4.6 discusses the potential application of the Ozturk pdf approximation algorithm to spread spectrum communication problems. Section 4.7 summarizes the conclusions about the use of SIRPs for interference modeling in communication applications.

4.2 Spherically Invariant Random Processes

Recent research by [1] on the theory of spherically invariant random processes (SIRPs) has proven very useful in modeling the joint probability density function of correlated, non-Gaussian radar clutter samples. SIRPs are closely related to Gaussian processes and exhibit many of the same characteristics, which is why they are such useful models. This section provides a brief summary of SIRP properties which are relevant to the detection problem for communication systems. Proofs and more detailed discussion of these properties, as well as several others, can be found in [1].

4.2.1 Definition of SIRPs

Let $X_i = x(t_i)$ denote a sample of the random process, $x(t)$, taken at time t_i. A random process, $x(t)$, is spherically invariant if for every N and every set of sampling times, t_i, $i = 1 \ldots N$, each vector, $\boldsymbol{X} = [X_1, X_2, \ldots, X_N]^T$, of samples of the random process has a pdf of the form

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = (2\pi)^{-\frac{N}{2}}|\Sigma|^{-\frac{1}{2}}h_N(p), \tag{4.1}$$

where p is the quadratic form

$$p = (\boldsymbol{x} - \boldsymbol{m}_x)^T\Sigma^{-1}(\boldsymbol{x} - \boldsymbol{m}_x). \tag{4.2}$$

Here, Σ and $\boldsymbol{m}_x$ are the $N \times N$ non-negative definite covariance matrix and mean vector of $\boldsymbol{x}$, respectively, and $h_N(\cdot)$ is a positive, real valued, monotonic decreasing function. A random vector, $\boldsymbol{x}$, with density function given by (4.1) is called a spherically invariant random vector (SIRV).

The above definition of an SIRP is analogous to the definition of a Gaussian random process, and in fact, substitution of

$$h_N(p) = \exp(-\frac{p}{2}) \tag{4.3}$$

into (4.1) results in a multivariate Gaussian density function. Therefore, one example of an SIRP is a Gaussian random process.

Observe that while $\boldsymbol{x}$ is denoted a *spherically* invariant random vector, the constant contours of $f_{\boldsymbol{X}}(\boldsymbol{x})$ are *ellipsoidal.* Constant spherical contours occur for $\Sigma = \boldsymbol{I}$, in which case $\boldsymbol{x}$ is called a spherically symmetric random vector (SSRV).

4.2.2 The Representation Theorem

The representation theorem for SIRVs states that if $\boldsymbol{X}$ is an N-dimensional, zero mean SIRV, then it can be written as the product,

$$\boldsymbol{X} = S\boldsymbol{Z}, \tag{4.4}$$

of an N-dimensional, zero mean, Gaussian random vector, $\boldsymbol{Z}$, and an independent, non-negative random variable, S, with a probability density function, $f_S(s)$, which is called the characteristic pdf of the SIRV. The pdf of any SIRV is uniquely determined by specifying its covariance matrix, mean vector, and characteristic pdf.

This theorem can be used to obtain an expression for the monotonic decreasing function, $h_N(p)$. From (4.4) it is seen that the random vector, $\boldsymbol{X}|(S = s)$, is conditionally Gaussian with covariance matrix, $s^2\Sigma_z$. This leads to the expression

$$h_N(p) = \int_0^\infty s^{-N} \exp(-\frac{p}{2s^2}) f_S(s)\, ds. \tag{4.5}$$

For convenience, and without loss of generality, the characteristic pdf is usually scaled to have unit mean squared value, i.e., $\mathrm{E}(S^2) = 1$. The special case $f_S(s) = \delta(s-1)$, where $\delta(\cdot)$ is the Dirac impulse function, again leads to (4.3) and a Gaussian density function.

The representation theorem provides one way of conceptualizing an SIRV. It is a zero mean, Gaussian random vector which has undergone randomization of a scale parameter. This has a physical interpretation for radar clutter modeling. The samples received from a particular range cell of some clutter region are jointly Gaussian distributed, but the average energy level from one range cell to another within the region may also be a random variable. This changing energy level corresponds to S in (4.4). Thus, a vector sample from any arbitrary range cell within the clutter region is an SIRV.

4.2.3 Linear Transformation Property

The linear transformation of an SIRV results in another SIRV with the same characteristic pdf. Specifically, if $\boldsymbol{X}$ is an N-dimensional SIRV with covariance matrix Σ_x, mean vector $\boldsymbol{m}_x$, and characteristic pdf, $f_S(s)$, then the vector, $\boldsymbol{Y}$, defined by the linear transformation,

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{X} + \boldsymbol{b}, \tag{4.6}$$

is also an SIRV with the same characteristic pdf, mean vector

$$\boldsymbol{m}_y = \boldsymbol{A}\boldsymbol{m}_x + \boldsymbol{b}, \tag{4.7}$$

and covariance matrix,

$$\Sigma_y = \boldsymbol{A}\Sigma_x\boldsymbol{A}^T. \tag{4.8}$$

Hence, the class of SIRVs is closed under linear transformations, including transformations to lower dimensions. One consequence of this result is that any subvector of an SIRV is also an SIRV with the same characteristic pdf as the full vector. This results in a simple determination of any marginal pdf of the SIRV. Commonly, the first or second order marginal pdf is used to classify the type of SIRV, e.g., K-distributed SIRV.

4.2.4 Additive Closure of SIRVs

The sum of two arbitrary, statistically independent SIRVs, say $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$, of the same dimension is not necessarily another SIRV. Hence, the class of SIRVs is not closed under addition. Two occasions when the sum, $\boldsymbol{X} = \boldsymbol{X}_1 + \boldsymbol{X}_2$, yields another SIRV are:

1. $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ are both Gaussian random vectors.

2. The covariance matrices, Σ_{X_1} and Σ_{X_2}, are related by $\Sigma_{X_1} = k\Sigma_{X_2}$, where k is any positive constant. The characteristic pdf of each SIRV may be different.

4.2.5 The Bootstrap Property

The higher order pdfs associated with an SIRP may be obtained from the recursive relations

$$h_{2N+1}(p) = (-2)^N \frac{d^N h_1(p)}{d\,p^N} \tag{4.9}$$

$$h_{2N+2}(p) = (-2)^N \frac{d^N h_2(p)}{d\,p^N}. \tag{4.10}$$

Since $h_N(p)$ must be positive for any N in order to have a valid N^{th} order pdf, the above equations indicate that the derivatives of $h_N(p)$ must alternate between negative, monotonic increasing and positive, monotonic decreasing functions. Hence, $h_N(p)$ must be a positive, monotonic decreasing function.

4.2.6 PDF of the Quadratic Form

The pdf of the quadratic form, $p = (\boldsymbol{x} - \boldsymbol{m}_x)^T \Sigma^{-1} (\boldsymbol{x} - \boldsymbol{m}_x)$, is

$$f_P(p) = \frac{p^{\frac{N}{2}-1}}{2^{\frac{N}{2}}\Gamma(\frac{N}{2})} h_N(p). \tag{4.11}$$

Since $h_N(p)$ is unique for each type of SIRV, the multivariate density function for a particular type of SIRV can be uniquely determined based upon the univariate density function of its quadratic form. This property significantly reduces the complexity of the pdf approximation that must be performed for optimal non-Gaussian processing.

4.2.7 Complex SIRVs

Let $\tilde{\boldsymbol{X}}$ denote a vector of N complex samples. The pdf of a complex SIRV, $\tilde{\boldsymbol{X}}$, can be derived by considering the representative form, $\tilde{\boldsymbol{X}} = S\tilde{\boldsymbol{Z}}$, where $\tilde{\boldsymbol{Z}}$ is a zero mean complex Gaussian random vector with complex covariance matrix, $\boldsymbol{\Sigma}$, and S is an independent, non-negative random variable with pdf, $f_S(s)$. Then the probability density function of $\tilde{\boldsymbol{X}}$ is given by

$$f_{\tilde{\boldsymbol{X}}}(\tilde{\boldsymbol{x}}) = \pi^{-N}|\Sigma|^{-1}h_N(\tilde{\boldsymbol{x}}^H\Sigma^{-1}\tilde{\boldsymbol{x}}), \tag{4.12}$$

where $\tilde{\boldsymbol{x}}^H$ denotes the Hermitian transpose of $\tilde{\boldsymbol{x}}$ and

$$h_N(p) = \int_0^\infty s^{-2N}\exp(-\frac{p}{s^2})f_S(s)\,ds \tag{4.13}$$

is a monotonic decreasing function of the quadratic form, $p = \tilde{\boldsymbol{x}}^H\Sigma^{-1}\tilde{\boldsymbol{x}}$.

4.2.8 Other Properties

Unimodality

Since $h_N(p)$ is a positive, monotonic decreasing function of the quadratic form, p, and p also describes the elliptical contours of constant density for any SIRV, it is obvious from (4.2) that the pdf of all SIRVs is unimodal. The mode, or peak, occurs at the mean value of the SIRV.

Statistical Independence

If the components of an SIRV are statistically independent, then that SIRV must be Gaussian. It is not possible for any other types of SIRVs to have independent components.

Ergodicity

Only Gaussian SIRPs are ergodic, as can be seen by considering the representation theorem of (4.4). Different sample functions of the SIRP, $\boldsymbol{X}$, correspond to different realizations of the random scalar, S. Evaluation of a quantity, such as the correlation function, by using the time average of a single sample function can yield a different answer for each sample function. This is because the sample functions are scaled differently. The time averages are the same only when S is a nonrandom constant, as is the case only for Gaussian SIRPs.

4.2.9 Examples

Some univariate probability density functions which have multivariate generalizations corresponding to SIRVs include Weibull, Student-t, chi, and K-distributed. Specific expressions for these and other SIRV density functions are given in [1].

4.3 Locally Optimum Detectors

Locally optimum detectors (LODs) have been extensively studied for applications to weak signal detection [13]. However, as pointed out by [2], most developments in the theory of LODs are based on the assumption of statistically independent samples and concentrate on showing that these suboptimum detectors are asymptotically optimum as the number of samples becomes very large. Large numbers of independent samples are usually not available for processing in most radar systems and while communication systems may have many more samples available, there are often instances when these samples are not statistically independent. Furthermore, the time required to accumulate a sufficiently large number of samples may conflict with any stationarity assumptions.

This section briefly reviews the development of LODs, particularly for weak signal detection in the presence of spherically invariant interference. A more detailed development, as well as some radar performance results in the presence of K-distributed and Student-t distributed interference, can be found in [2].

4.3.1 The Weak Signal Detection Problem

Consider a simple binary detection problem which must decide between two hypotheses, H_0 and H_1, given by

$$\begin{aligned} H_0 &: \ \boldsymbol{R} = \boldsymbol{N} \\ H_1 &: \ \boldsymbol{R} = \theta\boldsymbol{S} + \boldsymbol{N}, \end{aligned} \tag{4.3.1}$$

where $\boldsymbol{R} = [R_1, R_2, \ldots, R_N]^T$ is the received data, $\boldsymbol{S}$ is the transmitted signal, and $\boldsymbol{N}$ is the interference. The parameter, θ, is assumed to be an unknown, constant amplitude. The signal vector and noise process are assumed to be normalized so that θ^2 is a measure of the signal-to-interference ratio. The optimum receiver is known to be the likelihood ratio test (LRT),

$$\Lambda(\boldsymbol{r}) = \frac{f_{\boldsymbol{R}|H_1}(\boldsymbol{r}|H_1)}{f_{\boldsymbol{R}|H_0}(\boldsymbol{r}|H_0)} \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{4.3.2}$$

The conditional probability density functions can be expressed in terms of the noise pdf, $f_{\boldsymbol{N}}(\boldsymbol{n})$, which results in the LRT,

$$\Lambda(\boldsymbol{r}) = \frac{f_{\boldsymbol{N}}(\boldsymbol{r} - \theta\boldsymbol{s})}{f_{\boldsymbol{N}}(\boldsymbol{r})} \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{4.3.3}$$

For very small θ approaching zero, there is very little difference between the probability density functions under each hypothesis. Since the numerator and denominator of (4.3.3) are approximately equal, the likelihood ratio is always very close to unity. Thus, the optimum receiver of (4.3.3) may require a high degree of mathematical precision in calculation of the LRT if near optimum performance is to be obtained. Furthermore, the optimum receiver of (4.3.3) for a non-Gaussian noise density generally depends on the signal amplitude, θ. In such instances, unlike the Gaussian case, the optimum receiver is no longer uniformly most powerful (UMP) and cannot be exactly realized without knowledge of the signal amplitude. The LOD is a suboptimum receiver for weak signals ($\theta \approx 0$) which is approximately UMP.

4.3.2 Derivation of the LOD

Two derivations of the LOD receiver are given in [2]. One technique uses a Taylor series expansion and the other uses the method of Lagrange multipliers. A brief summary of these two different derivations is presented. Both developments follow [2] in focusing on the radar detection problem. However, as discussed in a later section, they can be applied to the communication problem with slight modifications.

Taylor Series Expansion Approach

The series expansion approach to the derivation of the LOD proceeds by expanding the numerator of (4.3.3) in a Taylor series about the received vector, $\boldsymbol{r}$. This results in

$$f_{\boldsymbol{N}}(\boldsymbol{r}-\theta\boldsymbol{s}) = f_{\boldsymbol{N}}(\boldsymbol{r}) + \sum_{m=1}^{\infty}(-1)^m\frac{\theta^m}{m!}[\boldsymbol{s}^T\nabla_{\boldsymbol{r}}]^m f_{\boldsymbol{N}}(\boldsymbol{r}), \tag{4.3.4}$$

where the operator, $\boldsymbol{s}^T\nabla_{\boldsymbol{r}}$, is defined to be

$$[\boldsymbol{s}^T\nabla_{\boldsymbol{r}}] = \sum_{k=1}^{N} s_k\frac{\partial}{\partial r_k}. \tag{4.3.5}$$

All the indicated partial derivatives are assumed to exist. Substituting (4.3.4) into (4.3.3) for the LRT yields

$$\Lambda(\boldsymbol{r}) = 1 + \sum_{m=1}^{\infty}(-1)^m\frac{\theta^m}{m!}\frac{[\boldsymbol{s}^T\nabla_{\boldsymbol{r}}]^m f_{\boldsymbol{N}}(\boldsymbol{r})}{f_{\boldsymbol{N}}(\boldsymbol{r})} \underset{H_0}{\overset{H_1}{\gtrless}} \eta. \tag{4.3.6}$$

Moving the unity constant into the threshold results in a new form of the LRT given by

$$\breve{\Lambda}(\boldsymbol{r}) = \sum_{m=1}^{\infty}(-1)^m\frac{\theta^m}{m!}\frac{[\boldsymbol{s}^T\nabla_{\boldsymbol{r}}]^m f_{\boldsymbol{N}}(\boldsymbol{r})}{f_{\boldsymbol{N}}(\boldsymbol{r})} \underset{H_0}{\overset{H_1}{\gtrless}} \eta - 1. \tag{4.3.7}$$

It was previously mentioned that the likelihood ratio, $\Lambda(\boldsymbol{r})$, is always near unity for the weak signal problem. However, the first term of the Taylor series expansion of $\Lambda(\boldsymbol{r})$ is also unity and when this value is subtracted from the threshold, the resulting *optimum* test of (4.3.7) can be more sensitive to a small variation in the test statistic.

The problem is now to evaluate the optimum statistic, $\check{\Lambda}(\boldsymbol{r})$. The LOD receiver is obtained for very small values of θ by assuming that the higher order terms of the summation in (4.3.7) are negligible, such that the first order term provides a good approximation to the optimum test statistic. This results in the approximate test

$$-\theta\frac{[\boldsymbol{s}^T\nabla_{\boldsymbol{r}}]f_{\boldsymbol{N}}(\boldsymbol{r})}{f_{\boldsymbol{N}}(\boldsymbol{r})} \underset{H_0}{\overset{H_1}{\gtrless}} \eta - 1. \tag{4.3.8}$$

Although θ is unknown, it is still a constant and can be moved into the threshold of (4.3.8). This gives a final general expression for the LOD test statistic as

$$T_{LOD}(\boldsymbol{r}) = \frac{[\boldsymbol{s}^T\nabla_{\boldsymbol{r}}]f_{\boldsymbol{N}}(\boldsymbol{r})}{f_{\boldsymbol{N}}(\boldsymbol{r})} \underset{H_0}{\overset{H_1}{\gtrless}} \frac{\eta-1}{\theta} = \eta'. \tag{4.3.9}$$

The single biggest advantage of the LOD is that the test of (4.3.9) can be completely determined without knowledge of the unknown parameter, θ. Strictly speaking, the LOD is not a UMP test because it is not the optimum test for every value of θ. However, it is very nearly optimal and for that reason is sometimes described as a *locally* most powerful test. In radar applications the threshold η' can be determined based upon a false alarm specification and, as shown later, η' is zero for a typical communication system.

Lagrange Multiplier Approach

For small values of the signal-to-interference-ratio (SIR), corresponding to $\theta \approx 0$ in (4.3.1), the probability of detection, P_D, is nearly equal to the probability of false alarm, P_{FA}. This is again due to the fact that the conditional pdf of the received signal is nearly the same for each hypothesis.

Traditional development of the optimum receiver uses a Neyman-Pearson strategy which maximizes the probability of detection subject to a specified probability of false alarm constraint. Considering the above discussion, though, it can be seen that the false alarm constraint limits the detection performance of any receiver for values of SIR near zero.

The strategy employed to develop an approximately UMP weak signal detector is to maximize the slope of the power function (P_D vs. θ) at the origin, subject to the false alarm constraint. The idea is that the detector with the largest such slope will attain higher detection probabilities much faster than other detectors within some one-sided range of θ, $0 < \theta < \theta^*$, near the origin. Such a detector would be the preferred one to use for that range of θ. The maximization procedure is performed using the method of Lagrange multipliers.

For the hypothesis testing problem of (4.3.1), define the nonrandomized decision rule

$$\phi(\boldsymbol{r}) = \begin{cases} 1; & H_1 \textit{ is true} \\ 0; & H_0 \textit{ is true} \end{cases}. \tag{4.3.10}$$

Then the detection and false alarm probablilities can be written as

$$P_D = Pr[\phi(\boldsymbol{r}) = 1|H_1] = \beta(\theta) = \int_{-\infty}^{\infty} \phi(\boldsymbol{r}) f_{\boldsymbol{R}|H_1}(\boldsymbol{r}|H_1)\, d\boldsymbol{r} \tag{4.3.11}$$

and

$$P_{FA} = Pr[\phi(\boldsymbol{r}) = 1|H_0] = \alpha = \int_{-\infty}^{\infty} \phi(\boldsymbol{r}) f_{\boldsymbol{R}|H_0}(\boldsymbol{r}|H_0)\, d\boldsymbol{r}, \tag{4.3.12}$$

where $\beta(\theta) = P_D$ denotes the power function of the test and $\alpha = P_{FA}$ is the significance level of the test. The aim is to maximize the slope of (4.3.11) at $\theta = 0$, subject to the constraint of (4.3.12). Formulating the maximization problem using the Lagrange multiplier approach gives

$$\begin{aligned} & max \left[\left[\frac{\partial}{\partial\theta} \int_{-\infty}^{\infty} \phi(\boldsymbol{r}) f_{\boldsymbol{R}|H_1}(\boldsymbol{r}|H_1)\, d\boldsymbol{r} \right] \Bigg|_{\theta=0} + \eta \left[\alpha - \int_{-\infty}^{\infty} \phi(\boldsymbol{r}) f_{\boldsymbol{R}|H_0}(\boldsymbol{r}|H_0)\, d\boldsymbol{r} \right] \right] \\ = \; & max \left[\int_{-\infty}^{\infty} \phi(\boldsymbol{r}) \left[\frac{\partial f_{\boldsymbol{R}|H_1}(\boldsymbol{r}|H_1)}{\partial\theta} - \eta f_{\boldsymbol{R}|H_0}(\boldsymbol{r}|H_0) \right] d\boldsymbol{r} \Bigg|_{\theta=0} \right] + \eta\alpha, \end{aligned} \tag{4.3.13}$$

where η is the Lagrange multiplier. The integral on the right side of (4.3.13) is maximized when the decision regions are chosen such that the integrand is always positive. This leads to the test

$$\frac{\left.\dfrac{\partial f_{\boldsymbol{R}|H_1}(\boldsymbol{r}|H_1)}{\partial\theta}\right|_{\theta=0}}{f_{\boldsymbol{R}|H_0}(\boldsymbol{r}|H_0)} = \frac{\left.\dfrac{\partial f_{\boldsymbol{N}}(\boldsymbol{r}-\theta\boldsymbol{s})}{\partial\theta}\right|_{\theta=0}}{f_{\boldsymbol{N}}(\boldsymbol{r})} \overset{H_1}{\underset{H_0}{\gtrless}} \eta. \tag{4.3.14}$$

This is obviously equivalent to (4.3.9), derived using the Taylor series expansion, when the relation

$$\left.\frac{\partial f_{\boldsymbol{N}}(\boldsymbol{r}-\theta\boldsymbol{s})}{\partial\theta}\right|_{\theta=0} = -\sum_{k=1}^{N} \frac{\partial f_{\boldsymbol{N}}(\boldsymbol{r})}{\partial r_k} s_k = -[\boldsymbol{s}^T \nabla_{\boldsymbol{r}}] f_{\boldsymbol{N}}(\boldsymbol{r}) \tag{4.3.15}$$

is substituted into (4.3.14). Thus, another expression for the LOD test statistic, based on the method of Lagrange multipliers, is

$$T_{LOD}(\boldsymbol{r}) = \frac{\left.\dfrac{\partial f_{\boldsymbol{N}}(\boldsymbol{r}-\theta\boldsymbol{s})}{\partial\theta}\right|_{\theta=0}}{f_{\boldsymbol{N}}(\boldsymbol{r})}. \tag{4.3.16}$$

4.3.3 LODs For SIRV Interference

The interference in the hypothesis testing problem of (4.3.1) is assumed to be of the form, $\boldsymbol{N} = \boldsymbol{C} + \boldsymbol{W}$, where $\boldsymbol{C}$ is a dominant SIRV interference typically associated with clutter, and $\boldsymbol{W}$ is an independent, white Gaussian noise component corresponding to the thermal noise level of the receiver. The clutter power is assumed to be much larger than the thermal noise level so that the total interference can be approximated by an SIRV with neglible error. The presence of the white, Gaussian noise component contributes a small amount to the diagonal elements of the total covariance matrix. This ensures a nonsingular covariance matrix and that detection performance is limited by the receiver thermal noise level.

Substitution of a general N-dimensional, zero mean, SIRV density function for $\boldsymbol{N}$,

$$f_{\boldsymbol{N}}(\boldsymbol{n}) = (2\pi)^{-\frac{N}{2}} |\Sigma|^{-\frac{1}{2}} h_N(\boldsymbol{n}^T \Sigma^{-1} \boldsymbol{n}), \tag{4.3.17}$$

into (4.3.16) for the LOD test statistic, $T_{LOD}(\boldsymbol{r})$, yields

$$T_{LOD}(\boldsymbol{r}) = -2(\boldsymbol{s}^T \Sigma^{-1} \boldsymbol{r}) \frac{h'_N(p)}{h_N(p)}, \tag{4.3.18}$$

where $h_N(\cdot)$ is a monotonic decreasing function of the quadratic form, $p = \boldsymbol{r}^T \Sigma^{-1} \boldsymbol{r}$. The LOD for detection of an unknown, constant amplitude signal in SIRV interference is a linear matched filter multiplied by a nonlinear function of the quadratic form.

For a Gaussian interference density with $h_N(p) = \exp(-p/2)$, the nonlinearity in (4.3.18) becomes a constant,

$$\frac{h'_N(p)}{h_N(p)} = -\frac{1}{2}. \tag{4.3.19}$$

Thus, the LOD test statistic reduces to

$$T_{LOD}(\boldsymbol{r}) = \boldsymbol{s}^T \Sigma^{-1} \boldsymbol{r}, \tag{4.3.20}$$

the optimum, linear matched filter for detection of an unknown, constant amplitude signal in Gaussian interference.

4.3.4 Additional Comments

Additional results for LODs, including cases of statistically independent components in the received vector, $\boldsymbol{r}$, and a random signal vector, $\boldsymbol{s}$, are included in [2]. The following sections on the use of SIRVs for modeling interference in the communication problem focus more on optimum detectors than on LODs. However, some relevance to LODs is presented in these sections.

4.4 Interference in Communication Systems

This section discusses the nature of non-Gaussian interference of interest in communication systems. Intentional jamming of spread-spectrum systems is of particular concern, though unintentional man-made sources and natural sources may also contribute to the interference environment. Most jamming waveforms have a constant envelope (CE) characteristic, due primarily to limitations on amplifier efficiency and peak power. This is a distinct example of non-Gaussian interference. Interference which is the sum of outputs from multiple, independent CE-jammers does not have a constant envelope characteristic and can also be very non-Gaussian.

The multiple jammer interference under consideration is assumed to occupy the same bandwidth as the received signal. If the system has some type of adaptive array capability, then these jammers are assumed to be entering the receiver through the main beam of the antenna so as to make adaptive processing ineffective. Non-Gaussian processing is most effective over conventional linear processing in situations where the receiver cannot filter out the interference without also removing significant signal energy.

4.4.1 Jammer Waveform Models

The multiple jammer interference is modeled by [8] in terms of in-phase and quadrature component samples as

$$I_k = \sum_{i=1}^{N_J} A_i \cos(\theta_{ik}) + n_{Ik},\ k = 1, \ldots, N \tag{4.4.1}$$

and

$$Q_k = \sum_{i=1}^{N_J} A_i \sin(\theta_{ik}) + n_{Qk},\ k = 1, \ldots, N, \tag{4.4.2}$$

where N_J is the number of jammers, A_i is the amplitude of the i^{th} jammer, and θ_{ik} is a random phase sequence of samples taken at times, t_k, $k = 1, \ldots, N$, for the i^{th} jammer. The terms, n_{Ik} and n_{Qk}, are jointly Gaussian thermal noise components. Appropriate selection of the random phase sequence in (4.4.1) and (4.4.2) allows representation of many types of jamming, including both wideband and narrowband by noise, M-ary phase shift keying, and simple direct noise amplification jammers.

The phase sequence model used by [6] is

$$\theta_{ik} = \psi + K_2 + K_3 k, \tag{4.4.3}$$

where K_2 is a constant offset, $K_3 t_k$ is a ramped phase component, and ψ is a random phase variable which is uniformly distributed between $\pm K_1$. A wide variety of jammer phase sequences can be created by this model. Several interesting histogram plots of the joint density function of the I_k and Q_k components for different jammer scenarios are given in [6]. In particular, when the phase sequence of (4.4.3) contains only the random phase term with $K_1 < 180°$, the joint pdf loses its radial symmetry. This loss of radial symmetry can happen for other jammer combinations, too.

For situations in which radial symmetry is present, the envelope (or amplitude) probability density function, $f_R(r)$, is related to the function, $f_{rcs}(r)$, describing the radial cross section of the joint pdf of the I and Q components by

$$f_R(r) = 2\pi r f_{rcs}(r), \tag{4.4.4}$$

where $r = \sqrt{I^2 + Q^2}$ is the amplitude or radial distance. A simple example is given by a radially symmetric, jointly bivariate, zero mean, Gaussian density function. The amplitude pdf is Rayleigh distributed, which is zero at the origin, and the radial cross section function is a decaying exponential of the form $\exp(-r^2)$, which is maximum at the origin. The difference between the radial cross section function and the amplitude pdf has been pointed out to prevent confusion when discussing nonunimodal probability density functions.

The amplitude pdf, $f_R(r)$, for the sum of thermal noise and N_J independent CE-jammers with equal amplitudes and radial symmetry is illustrated by the histogram plots in Figure 4.4.1 for $N_J = 1, \ldots, 5$. The total jammer power in each case is fixed at 33dB above the average thermal noise power. This sequence of plots show one immediate problem of countering multiple jammers. The pdf of the sum of independent CE-jammers with approximately equal amplitudes tends to the Gaussian density by the central limit theorem. This explains why the amplitude pdf for $N_J = 5$ in Figure 4.4.1 appears similar to the Rayleigh density function. The possible performance gains of optimal non-Gaussian processing are decreased as the amplitude pdf approaches the Rayleigh pdf, which begins to happen very quickly with sums of equal amplitude jammers.

Figure 4.4.2 shows histogram plots that indicate features of the radial cross section functions of the joint pdf of I and Q components for the same jamming scenarios as Figure 4.4.1. The horizontal axis of these plots has been distorted but the vertical axis has not, which doesn't change the nonunimodal property of the $N_J = 2$ case. There is a peak at the origin and a smaller peak at another radial distance from the origin. Observe that the nonunimodal property of the joint pdf of this case cannot be determined from its amplitude pdf given in Figure 4.4.1. This nonunimodal behavior is also pointed out in [6], with $K_1 = 180°$ in the random phase model of (4.4.3), and in [8].

4.4.2 Ability of SIRPs to Model Jammer Waveforms

The previous section raises several points which indicate it may not be appropriate to use SIRPs in modeling CE-jammer waverforms. A joint pdf model for the general case of N pairs of I-Q data would be desired to have univariate marginal

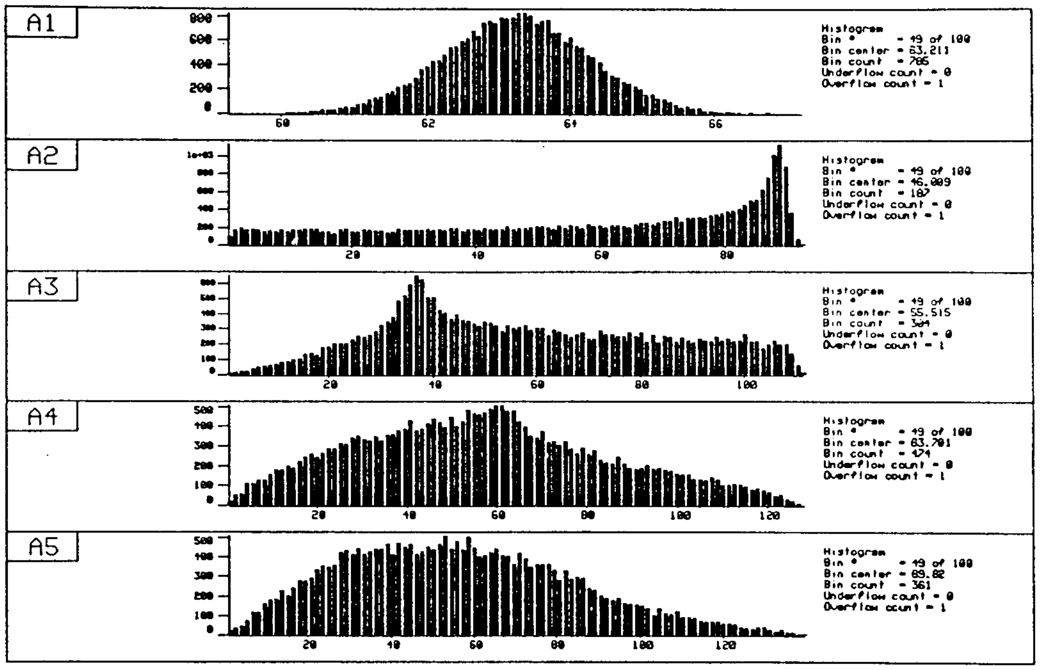


Figure 4.4.1: Amplitude pdf of the sum of N_J equal amplitude jammers – *A1)* $N_J = 1$; *A2)* $N_J = 2$; *A3)* $N_J = 3$; *A4)* $N_J = 4$; *A5)* $N_J = 5$.

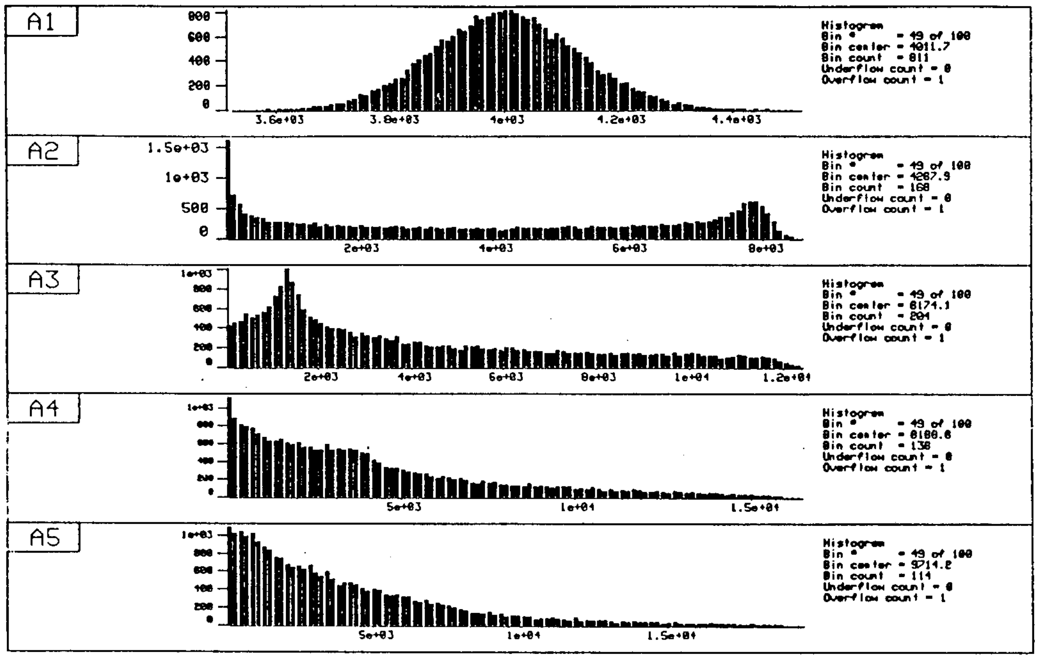


Figure 4.4.2: Radial cross section function for the joint pdf of *I*-*Q* components of the sum of N_J equal amplitude jammers – *A1)* $N_J = 1$; *A2)* $N_J = 2$; *A3)* $N_J = 3$; *A4)* $N_J = 4$; *A5)* $N_J = 5$.

densities which have some of the characteristics described above. However, some of these properties conflict with characteristics of SIRVs.

First, the bivariate marginal density of any SSRV must be radially symmetric. Thus, radial symmetry corresponds to spherical symmetry for the multivariate pdf and generalization of the bivariate pdfs which do not have radial symmetry is not possible with SSRVs.

Next, as indicated in Section 4.2.8, SIRVs must be unimodal with the peak in the pdf occurring at the mean vector. This property follows because the radial cross section function of the bivariate joint pdf of I and Q is directly proportional to $h_N(p)$, which is monotonic decreasing. The sum of two nearly equal amplitude jammers is a scenario of frequent interest which has a nonunimodal joint pdf, as seen above. A multivariate SSRV model with this property does not exist.

Finally, the jammer model of (4.4.1) and (4.4.2) raises a concern regarding the lack of an additive closure property for SIRVs discussed in Section 4.2.4. For example, if it is assumed an SIRV can model the interference pdf of a single jammer for certain types of jammers, then the sum of N_J of these jammers is an SIRV only if their covariance matrices are constant multiples of a common covariance matrix, i.e., $\Sigma_i = k_i\Sigma, i = 1, \ldots, N_J$. This is a very restrictive condition on independent jammers.

The best chance of this condition being satisfied for multiple jammers probably occurs when samples from the output of each individual jammer in the sum are uncorrelated. This can occur if the phase sequence of each jammer is a random variable, uniformly distributed on $(0, 2\pi)$. However, results in [18] show that samples of this random process are not spherically invariant, even though the joint pdf is radially symmetric. Note that the use of the term spherically symmetric by [17, 18] is different than the usage here and does not imply the vectors are spherically invariant.

In summary, there are several problems which restrict, and perhaps eliminate, consideration of SIRPs as models for jammer waveforms in spread spectrum communication problems. There is a more general class of spherical distributions [15, 16], of which SIRVs are a special case, which may be a source of possible models. However, some of the appealing properties of SIRVs, such as the representation theorem and the bootstrap property, are lost. The next section proceeds with an investigation of the optimum non-Gaussian processor under the assumption that the interference of interest can be modeled as an SIRV.

4.5 M-ary Detection in SIRV Interference

A communication system which transmits one of M symbols is considered. Each symbol is represented by a different transmitted waveform, $s_i(t), i = 1, \ldots, M$. The system must typically operate in an environment that may consist of many types of intentional or unintentional jamming, as well as natural sources of interference. The receiver must determine which of the M symbols has been transmitted, based upon samples of a received waveform which are corrupted by this interference. This M-hypotheses decision problem is represented by

$$\begin{aligned} H_1 &: \boldsymbol{R} = ae^{j\phi}\boldsymbol{S}_1 + \boldsymbol{N} \\ H_2 &: \boldsymbol{R} = ae^{j\phi}\boldsymbol{S}_2 + \boldsymbol{N} \\ &\vdots \\ H_M &: \boldsymbol{R} = ae^{j\phi}\boldsymbol{S}_M + \boldsymbol{N}, \end{aligned} \tag{4.5.1}$$

where $\boldsymbol{R} = [R_1, \ldots, R_N]^T$ is the received data vector, $\boldsymbol{N} = [N_1, \ldots, N_N]^T$ is the additive interference vector, and $\boldsymbol{S}_i = [S_{i1}, \ldots, S_{iN}]^T, i = 1, \ldots, M$ are the transmitted signal vectors. The channel attenuation and possible uncertainty in the phase of the signal, due to either channel effects or uncertainty in the transmitter, are accounted for by the complex factor, $a\exp(j\phi)$.

The optimum receiver based upon a minimum probability of error criterion is one which selects the hypothesis with the maximum of the a posteriori probabilities [5],

$$Pr[H_i|\boldsymbol{R}] = \frac{P_i f_{\boldsymbol{R}|H_i}(\boldsymbol{r}|H_i)}{f_{\boldsymbol{R}}(\boldsymbol{r})}, \; i = 1, \ldots, M, \tag{4.5.2}$$

where P_i is the probability that the i^{th} symbol is transmitted. Since the pdf of $\boldsymbol{r}$ in the denominator of (4.5.2) is the same for each of the M hypotheses, an equivalent optimum receiver selects the symbol corresponding to the largest of the quantities $P_i f_{\boldsymbol{R}|H_i}(\boldsymbol{r}|H_i)$.

For communication systems an equiprobable occurence of each symbol is assumed, since this usually corresponds to maximum information content in the transmitted message. Under the assumption of $P_i = \frac{1}{M}, i = 1, \ldots, M$, the optimum receiver selects the symbol with the largest of the likelihood functions, $L_i(\boldsymbol{r})$, defined as

$$L_i(\boldsymbol{r}) = f_{\boldsymbol{R}|H_i}(\boldsymbol{r}|H_i), \; i = 1, \ldots, M. \tag{4.5.3}$$

It is assumed that the received signals are processed coherently in terms of inphase (I) and quadrature (Q) components. Then $\boldsymbol{R}, \boldsymbol{N}$, and $\boldsymbol{S}_i$ either represent vectors containing N complex samples of the lowpass complex envelope or $2N$ real samples of the I and Q components. Both representations lead to the same results. However, the following analysis uses real vectors when only the amplitude, a, is unknown and complex vectors whenever the phase, ϕ, is unknown. Henceforth, the use of complex lowpass envelope vectors is specifically indicated by the notation $\tilde{\boldsymbol{R}}, \tilde{\boldsymbol{N}}$, and $\tilde{\boldsymbol{S}}_i$.

Development of the optimum M-ary detector in SIRV interference depends on whether the unknown parameters, a and ϕ, are constants or random variables. Optimum receivers are presented for the cases: (i) unknown constant amplitude and known phase, (ii) random amplitude and known phase, and (iii) known amplitude and random phase. The optimum receivers for cases (i) and (ii) are shown to be the same as the optimum Gaussian receiver.

4.5.1 Unknown Constant Amplitude

In this case the amplitude parameter, a, is assumed to be an unknown constant while the phase parameter is assumed known and arbitrarily selected to be $\phi = 0$. The N-dimensional interference vector, $\boldsymbol{N}$, in (4.5.1) is assumed to be a zero mean SIRV with covariance matrix, $\boldsymbol{\Sigma}$. Then the likelihood functions of (4.5.3) expressed in terms of the probability density function of the interference,

$$f_{\boldsymbol{N}}(\boldsymbol{n}) = (2\pi)^{-\frac{N}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}}h_N(\boldsymbol{n}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{n}), \tag{4.5.4}$$

become

$$L_i(\boldsymbol{r}) = (2\pi)^{-\frac{N}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}}h_N[(\boldsymbol{r} - a\boldsymbol{s}_i)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_i)],\ i = 1, \ldots, M. \tag{4.5.5}$$

Since $h_N(\cdot)$ is a monotonic decreasing function, selecting the largest likelihood function in (4.5.5) is equivalent to selecting the smallest of the quadratic forms,

$$\begin{aligned} p_i &= (\boldsymbol{r} - a\boldsymbol{s}_i)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_i) \\ &= \boldsymbol{r}^T\boldsymbol{\Sigma}^{-1}\boldsymbol{r} - 2a\boldsymbol{s}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{r} + a^2\boldsymbol{s}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{s}_i. \end{aligned} \tag{4.5.6}$$

The first term of (4.5.6) is the same for all hypotheses. Hence, the optimum receiver selects the hypothesis having the greatest of the values,

$$T_i(\boldsymbol{r}) = 2\boldsymbol{s}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{r} - a\boldsymbol{s}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{s}_i. \tag{4.5.7}$$

This result is identical to the result for detection of a signal with unknown, constant amplitude in additive Gaussian noise. This can be recognized by considering that $h_N(p_i) = \exp(-p_i/2)$ for Gaussian noise has a logarithmic inverse function. Applying this inverse to the likelihood functions and selecting the maximum of the resultant log-likelihood functions leads directly to the results of (4.5.6) and (4.5.7). Some standard special cases are now presented to give further insight into this result.

Special Case 1: Equal Energy Signals.

If all transmitted signals have equal energy given by $E_s = \boldsymbol{s}_i^T\boldsymbol{s}_i$, for $1 \leq i \leq M$, then for white noise with covariance matrix, $\boldsymbol{\Sigma} = \boldsymbol{I}$, the statistic of (4.5.7) simplifies to

$$T_i(\boldsymbol{r}) = \boldsymbol{s}_i^T\boldsymbol{r}. \tag{4.5.8}$$

This is a simple discrete correlation operation and the receiver selects the hypothesis corresponding to the signal that is most correlated with the received vector. This is a uniformly most powerful test because it does not depend on any knowledge of the unknown amplitude.

However, if the noise is correlated, then the equal energy signal assumption does not imply that the $a\boldsymbol{s}_i^T\boldsymbol{\Sigma}^{-1}\boldsymbol{s}_i$ term of (4.5.7) is the same for each hypothesis. Hence, the resulting test depends on the value of a.

Special Case 2: Binary Detection.

Some additional insight may be gained by considering the case of binary detection ($M = 2$). The optimum Bayes' test for this case is the likelihood ratio test

$$\begin{aligned}\Lambda(\boldsymbol{r}) &= \frac{f_{\boldsymbol{R}|H_1}(\boldsymbol{r}|H_1)}{f_{\boldsymbol{R}|H_2}(\boldsymbol{r}|H_2)} = \frac{f_{\boldsymbol{N}}(\boldsymbol{r} - a\boldsymbol{s}_1)}{f_{\boldsymbol{N}}(\boldsymbol{r} - a\boldsymbol{s}_2)} \\ &= \frac{(2\pi)^{-\frac{N}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}}h_N[(\boldsymbol{r} - a\boldsymbol{s}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_1)]}{(2\pi)^{-\frac{N}{2}}|\boldsymbol{\Sigma}|^{-\frac{1}{2}}h_N[(\boldsymbol{r} - a\boldsymbol{s}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_2)]} \\ &= \frac{h_N[(\boldsymbol{r} - a\boldsymbol{s}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_1)]}{h_N[(\boldsymbol{r} - a\boldsymbol{s}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_2)]} \underset{H_2}{\overset{H_1}{\gtrless}} \eta. \end{aligned} \tag{4.5.9}$$

The Bayes' threshold, η, is given by

$$\eta = \frac{P_2(C_{12} - C_{22})}{P_1(C_{21} - C_{11})}, \tag{4.5.10}$$

where C_{ij} is the cost of selecting hypothesis i when hypothesis j is true.

The cost assignments for a minimum probability of error receiver are $C_{11} = C_{22} = 0$ and $C_{12} = C_{21} = 1$. The case of equal probability signals, $P_1 = P_2 = 1/2$, leads to $\eta = 1$ and the LRT becomes

$$\frac{h_N[(\boldsymbol{r} - a\boldsymbol{s}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_1)]}{h_N[(\boldsymbol{r} - a\boldsymbol{s}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_2)]} \underset{H_2}{\overset{H_1}{\gtrless}} 1, \tag{4.5.11}$$

or, equivalently,

$$h_N[(\boldsymbol{r} - a\boldsymbol{s}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_1)] \underset{H_2}{\overset{H_1}{\gtrless}} h_N[(\boldsymbol{r} - a\boldsymbol{s}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_2)]. \tag{4.5.12}$$

Since $h_N(\cdot)$ is a monotonic decreasing function, its inverse function, $h_N^{-1}(\cdot)$, is also monotonic decreasing and can be applied to both sides of (4.5.12). The resulting test,

$$(\boldsymbol{r} - a\boldsymbol{s}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_1) \underset{H_2}{\overset{H_1}{\lessgtr}} (\boldsymbol{r} - a\boldsymbol{s}_2)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{r} - a\boldsymbol{s}_2), \tag{4.5.13}$$

can be simplified to

$$(\boldsymbol{s}_2 - \boldsymbol{s}_1)^T\boldsymbol{\Sigma}^{-1}\boldsymbol{r} \underset{H_1}{\overset{H_2}{\gtrless}} \frac{a}{2}(\boldsymbol{s}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{s}_2 - \boldsymbol{s}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{s}_1), \tag{4.5.14}$$

independent of the type of SIRV interference. The inequalities in (4.5.13) are reversed because $h_N^{-1}(\cdot)$ is a decreasing function. The simplification of the LRT obtained by applying the inverse of $h_N(\cdot)$ is possible only because the Bayes' threshold is specifically determined to be unity in the communication problem. When the threshold has any other value, as is often the case for radar detection where the false alarm rate specification determines the threshold, the LRT involves a more complicated nonlinear function which depends upon the type of SIRV.

Special Case 3: Locally Optimum Detector.

It is interesting to compare the LOD to the optimum receiver for binary detection of equal energy signals in uncorrelated SIRV interference. The optimum receiver of (4.5.14) for this case is

$$(\boldsymbol{s}_2 - \boldsymbol{s}_1)^T \Sigma^{-1} \boldsymbol{r} \underset{H_1}{\overset{H_2}{\gtrless}} 0. \tag{4.5.15}$$

The LOD is derived by considering the optimum minimum probability of error receiver in the form

$$1 + T_{LOD}(\boldsymbol{r}) + (\textit{higher order terms}) \underset{H_1}{\overset{H_2}{\gtrless}} 1. \tag{4.5.16}$$

Neglecting the higher order terms yields

$$T_{LOD}(\boldsymbol{r}) \underset{H_1}{\overset{H_2}{\gtrless}} 0, \tag{4.5.17}$$

which becomes

$$\boldsymbol{s}^T \Sigma^{-1} \boldsymbol{r} \{-2 \frac{h'_N(p)}{h_N(p)}\} \underset{H_1}{\overset{H_2}{\gtrless}} 0 \tag{4.5.18}$$

when the general form of the LOD in (4.3.18) for SIRVs is used. The zero threshold makes T_{LOD} equivalent to a sign test. Since the nonlinear function enclosed in $\{\cdot\}$ is always positive for all SIRPs, it does not change the sign of the statistic and can be removed. The remaining test is exactly the same as the optimum test of (4.5.15).

4.5.2 Unknown Random Amplitude

The amplitude parameter, a, is assumed to be a random variable with probability density function, $f_A(a)$, and the phase is again arbitrarily assumed to be $\phi = 0$. The likelihood functions of (4.5.3) now become

$$\begin{aligned} L_i(\boldsymbol{r}) &= \int_0^\infty f_{\boldsymbol{R}|a,H_i}(\boldsymbol{r}|a, H_i) f_A(a)\, da \\ &= \int_0^\infty (2\pi)^{-\frac{N}{2}} |\boldsymbol{\Sigma}|^{-\frac{1}{2}} h_N[(\boldsymbol{r} - a\boldsymbol{s}_i)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{r} - a\boldsymbol{s}_i)] f_A(a)\, da \end{aligned} \tag{4.5.19}$$

for $i = 1, \ldots, M$. Substituting the integral expression of (4.5) for $h_N(\cdot)$ into the above expression yields

$$L_i(\boldsymbol{r}) = k_N \int_0^\infty |! \int_0^\infty s^{-N} \exp(-\frac{(\boldsymbol{r} - a\boldsymbol{s}_i)^T \boldsymbol{\Sigma}^{-1} (\boldsymbol{r} - a\boldsymbol{s}_i)}{2s^2}) f_S(s)\, ds f_A(a)\, da, \tag{4.5.20}$$

where the constant, k_N, has been defined as

$$k_N = (2\pi)^{-\frac{N}{2}}|\Sigma|^{-\frac{1}{2}}. \tag{4.5.21}$$

Rearranging the order of the integrations in (4.5.20) leads to

$$L_i(\boldsymbol{r}) = k_N \int_0^\infty s^{-N} f_S(s) \underbrace{\int_0^\infty \exp(-\frac{(\boldsymbol{r}-a\boldsymbol{s}_i)^T\Sigma^{-1}(\boldsymbol{r}-a\boldsymbol{s}_i)}{2s^2}) f_A(a)\, da}_{\Gamma_i(\boldsymbol{r},s)}\, ds. \tag{4.5.22}$$

The minimum probability of error receiver again chooses the hypothesis with the largest of the likelihood values given by (4.5.22). The conditions necessary for some particular hypothesis, H_j, to have the largest likelihood function can be obtained by comparing it to any other arbitrary hypothesis, say H_k.

First, for both H_j and H_k, consider the inner integral indicated by $\Gamma_i(\boldsymbol{r}, s)$ in (4.5.22). The condition on the random variable, s, for which

$$\Gamma_j(\boldsymbol{r}, s) > \Gamma_k(\boldsymbol{r}, s) \tag{4.5.23}$$

holds is determined by the inequality

$$\begin{aligned}
&\Gamma_j(\boldsymbol{r}, s) - \Gamma_k(\boldsymbol{r}, s) \\
&= \int_0^\infty \exp(-\frac{p_j}{2s^2}) f_A(a)\, da - \int_0^\infty \exp(-\frac{p_k}{2s^2}) f_A(a)\, da \\
&= \int_0^\infty [\exp(-\frac{p_j}{2s^2}) - \exp(-\frac{p_k}{2s^2})] f_A(a)\, da \;>\; 0, \qquad (4.5.24)
\end{aligned}$$

which indicates the integrand must be positive. Hence, it must satisfy

$$[\exp(-\frac{p_j}{2s^2}) - \exp(-\frac{p_k}{2s^2})] f_A(a) > 0. \tag{4.5.25}$$

Since $f_A(a)$ is nonnegative and the exponential is a monotonic increasing function, the condition of (4.5.25) is equivalent to

$$-\frac{p_j}{2s^2} > -\frac{p_k}{2s^2}, \tag{4.5.26}$$

This simplifies to a comparison of the quadratic forms,

$$p_j < p_k, \tag{4.5.27}$$

which does not depend on the random variable, S!

This result is very significant because, combined with the fact that S is nonnegative, it indicates that the outer integral with respect to s in (4.5.22) is irrelevant in determining which likelihood function is the largest. *Therefore, the optimum processor is independent of what type of SIRV is present and must be equivalent to the* *optimum Gaussian receiver*, since the class of SIRV density functions includes the Gaussian pdf.

Also, since the result in (4.5.27) does not depend on any knowledge of the density function, $f_A(a)$, then the unknown, constant amplitude case studied in Section 4.5.1 could be considered a special case with $f_A(a) = \delta(a - a_0)$, where a_0 is the unknown constant. This provides further verification of the result obtained there.

These two separate results from Section 4.5.1 and Section 4.5.2 can be combined into the following conclusion. The minimum probability of error receiver in SIRV interference for M equiprobable signals with an unknown amplitude is exactly the same as the optimum Gaussian receiver.

4.5.3 Unknown Random Phase

The development of the minimum probability of error communication receiver in non-Gaussian noise for the previous two cases involving unknown amplitude has led to results which are identical to the optimum Gaussian receiver. The amplitude is now assumed to be a known constant, a, while the phase is a random variable, ϕ, with probability density function,

$$f_\Phi(\phi) = \frac{1}{2\pi}, \quad -\pi \le \phi \le \pi. \tag{4.5.28}$$

This section briefly develops the general form of the optimum M-ary receiver and then gives an example of a closed form solution which does not reduce to the Gaussian receiver.

Optimum M-ary Receiver

The M-hypothesis problem of (4.5.1) can also be expressed in terms of the complex envelopes of the lowpass processes as

$$\begin{aligned} H_1 &: \tilde{\boldsymbol{R}} = ae^{j\phi}\tilde{\boldsymbol{S}}_1 + \tilde{\boldsymbol{N}} \\ H_2 &: \tilde{\boldsymbol{R}} = ae^{j\phi}\tilde{\boldsymbol{S}}_2 + \tilde{\boldsymbol{N}} \\ &\vdots \\ H_M &: \tilde{\boldsymbol{R}} = ae^{j\phi}\tilde{\boldsymbol{S}}_M + \tilde{\boldsymbol{N}}, \end{aligned} \tag{4.5.29}$$

where $\tilde{\boldsymbol{R}}$ is the lowpass complex envelope of the received signal, $\tilde{\boldsymbol{S}}_i$ is the lowpass complex envelope of the transmitted signal for the i^{th} hypothesis, and $\tilde{\boldsymbol{N}}$ is the lowpass complex envelope of the spherically invariant noise process. The minimum probability of error receiver selects the hypothesis with the largest of the likelihood functions,

$$L_i(\tilde{\boldsymbol{r}}) = \int_{-\pi}^{\pi} L_i(\tilde{\boldsymbol{r}}|\phi) f_\Phi(\phi)\, d\phi, \tag{4.5.30}$$

where the conditional likelihood functions, $L_i(\tilde{\boldsymbol{r}}|\phi)$, are defined to be

$$L_i(\tilde{\boldsymbol{r}}|\phi) = f_{\tilde{\boldsymbol{R}}|\phi,H_i}(\tilde{\boldsymbol{r}}|\phi, H_i) = f_{\tilde{\boldsymbol{N}}}(\tilde{\boldsymbol{r}} - ae^{-j\phi}\tilde{\boldsymbol{s}}_i) \tag{4.5.31}$$

for i=1,...,M. The density function of zero mean, complex SIRV noise with known covariance matrix, $\boldsymbol{\Sigma}$, and characteristic pdf, $f_S(s)$, is given in Section 4.2.7 as

$$f_{\tilde{\boldsymbol{N}}}(\tilde{\boldsymbol{n}}) = \pi^{-N}|\boldsymbol{\Sigma}|^{-1}h_N(\tilde{\boldsymbol{n}}^H\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{n}}). \tag{4.5.32}$$

Substituting this pdf and (4.5.31) into (4.5.30) gives

$$L_i(\tilde{\boldsymbol{r}}) = \int_{-\pi}^{\pi} \pi^{-N}|\boldsymbol{\Sigma}|^{-1}h_N[(\tilde{\boldsymbol{r}} - ae^{-j\phi}\tilde{\boldsymbol{s}}_i)^H\boldsymbol{\Sigma}^{-1}(\tilde{\boldsymbol{r}} - ae^{-j\phi}\tilde{\boldsymbol{s}}_i)]f_\Phi(\phi)\,d\phi. \tag{4.5.33}$$

The quadratic form in the argument of $h_N(\cdot)$ is real and can be expanded to

$$\begin{aligned}
&(\tilde{\boldsymbol{r}} - ae^{-j\phi}\tilde{\boldsymbol{s}}_i)^H\boldsymbol{\Sigma}^{-1}(\tilde{\boldsymbol{r}} - ae^{-j\phi}\tilde{\boldsymbol{s}}_i) \\
&\quad= \tilde{\boldsymbol{r}}^H\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{r}} + a^2\tilde{\boldsymbol{s}}_i^H\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{s}}_i - 2aRe\{e^{-j\phi}\tilde{\boldsymbol{s}}_i^H\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{r}}\} \\
&\quad= \tilde{\boldsymbol{r}}^H\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{r}} + a^2\tilde{\boldsymbol{s}}_i^H\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{s}}_i - 2a[L_{c_i}\cos(\phi) - L_{s_i}\sin(\phi)],
\end{aligned} \tag{4.5.34}$$

where L_{c_i} and L_{s_i} are obtained from the definition

$$\tilde{\boldsymbol{s}}_i^H\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{r}} = L_{c_i} + jL_{s_i}. \tag{4.5.35}$$

Substituting (4.5.34), the integral expression for $h_N(\cdot)$ given by (4.13) in Section 4.2.7, and the pdf for ϕ given by (4.5.28) into (4.5.33) yields the following equivalent likelihood statistic of the i^{th} hypothesis,

$$T_i(\tilde{\boldsymbol{r}}) = \mathcal{L}\left\{t^{N-\frac{3}{2}}f_S(t^{-\frac{1}{2}})I_0\left[a\sqrt{L_{c_i}^2 + L_{s_i}^2}\,t\right]\right\}, \tag{4.5.36}$$

where $I_0(\cdot)$ is a modified Bessel function of the first kind, and $\mathcal{L}\{\cdot\}$ is the one-sided Laplace transform operator defined by

$$\mathcal{L}\{g(t)\} = \int_0^\infty g(t)e^{-k(\tilde{\boldsymbol{r}})t}\,dt, \tag{4.5.37}$$

with the Laplace transform frequency variable, $k(\tilde{\boldsymbol{r}})$, defined by

$$k(\tilde{\boldsymbol{r}}) = \frac{1}{2}(\tilde{\boldsymbol{r}}^H\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{r}} + a^2\tilde{\boldsymbol{s}}_i^H\boldsymbol{\Sigma}^{-1}\tilde{\boldsymbol{s}}_i). \tag{4.5.38}$$

The derivation of the result in (4.5.36) is very long and has been omitted here. The minimum probability of error receiver selects the hypothesis which has the largest equivalent likelihood statistic, $T_i(\tilde{\boldsymbol{r}})$.

The result of (4.5.36) gives the correct result for the case of a Gaussian SIRV, which corresponds to evaluating the integrand of the Laplace transform integral at $t = 1$. This leads to the equivalent likelihood statistic for the Gaussian problem,

$$T_{G_i}(\tilde{\boldsymbol{r}}) = \exp(-\frac{a^2 \tilde{\boldsymbol{s}}_i^H \boldsymbol{\Sigma}^{-1} \tilde{\boldsymbol{s}}_i}{2}) I_0 \left[a\sqrt{L_{c_i}^2 + L_{s_i}^2} \right], \tag{4.5.39}$$

which can be verified by consulting [5] and [11].

It is not be possible to use (4.5.36) to obtain a closed form solution for every type of SIRV. However, a brief investigation discovered a closed form solution for the Student-t example, which is discussed next.

Student-t Example

The Student-t SIRV has the characteristic pdf [1],

$$f_S(s) = \frac{2b^{2\nu-1}}{2^\nu \Gamma(\nu)} s^{-(2\nu+1)} \exp(-\frac{b^2}{2s^2}), \ s \geq 0, \tag{4.5.40}$$

where ν is the shape parameter of the density function and b is a scale parameter. Substituting this characteristic pdf expression into (4.5.36) and using a table of Laplace transforms [12] leads to the equivalent likelihood statistic for the i^{th} hypothesis,

$$T_i(\tilde{\boldsymbol{r}}) = \left[\sqrt{(k(\tilde{\boldsymbol{r}}) + \frac{b^2}{2})^2 - a^2(L_{c_i}^2 + L_{s_i}^2)} \right]^{-(N+\nu)}$$

$$\times \ {}_2F_1 \left(1 - N - \nu, N + \nu; 1; \frac{1}{2} - \frac{k(\tilde{\boldsymbol{r}}) + \frac{b^2}{2}}{\left[2\sqrt{(k(\tilde{\boldsymbol{r}}) + \frac{b^2}{2})^2 - a^2(L_{c_i}^2 + L_{s_i}^2)} \right]} \right) \tag{4.5.41}$$

where ${}_2F_1(a, b; c; z)$ is the Gauss hypergeometric function, a is an assumed known amplitude parameter, and L_{c_i},L_{s_i}, and $k(\tilde{\boldsymbol{r}})$ are given by (4.5.35) and (4.5.38), respectively.

The minimum probability of error receiver in spherically invariant Student-t interference selects the hypothesis with the largest value of the statistic given by (4.5.41). This statistic cannot be reduced to that obtained for the optimum Gaussian receiver because it is not a monotonic function of the Gaussian receiver statistic, which is partially identified as

$$L_{c_i}^2 + L_{s_i}^2 = ||\tilde{\boldsymbol{s}}_i^H \boldsymbol{\Sigma}^{-1} \tilde{\boldsymbol{r}}||^2, \tag{4.5.42}$$

the familiar matched filter followed by envelope detection. The Gauss hypergeometric function, ${}_2F_1(a, b; c; z)$, reduces to a polynomial in z whenever either of the arguments a or b equals $-n$, for $n = 0, 1, 2, \ldots$. This occurs in (4.5.41) for integer

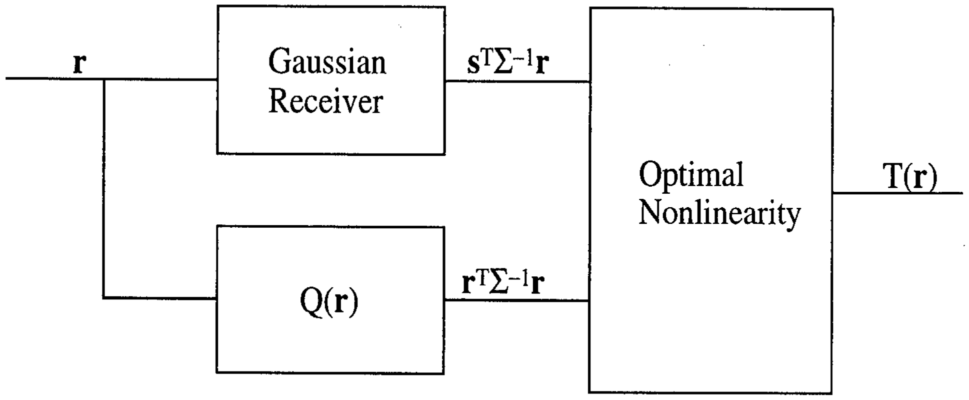


Figure 4.5.1: Canonical structure for optimal processors in SIRV interference.

values of the shape parameter, ν. The receiver in (4.5.42) is complete for equal energy transmitted signals in white Gaussian noise. However, in general, the quantity $\tilde{s}_i^H \Sigma^{-1} \tilde{s}_i$ is needed for the complete receiver, as seen by considering (4.5.39).

Comments

If the amplitude parameter, a, is also assumed to be an unknown random variable, then the optimum receiver could be obtained by integrating (4.5.41) over the probability density function of the amplitude. This situation may be representative of some types of fading channels.

The optimal receiver for detection in SIRV interference has a canonical structure, as shown in Figure 4.5.1, for all the cases that have been considered. This is also the structure of the LOD receiver given by (4.3.18). This canonical form is very significant because it allows the Gaussian receiver to be used as part of any optimal processor, which may minimize the impact on the structure of existing receivers.

4.6 The Ozturk Algorithm for PDF Approximation

The Ozturk algorithm is a very powerful technique used to select a univariate pdf approximation for a set of random data. It has the potential to improve upon typical methods of pdf approximation investigated for spread spectrum communication receivers. The mathematical details of the algorithm are found in [3, 4].

There are two steps performed by the Ozturk algorithm. First, it performs a goodness of fit test of the data set against a specified null distribution. This null distribution is arbitrary, but is usually assumed to be Gaussian. The second step of the algorithm selects a probability density function which best approximates the data set.

The goodness of fit procedure in the first step uses a reference distribution, not necessarily the same as the null distribution, and the ordered statistics of the data to generate a trajectory for the data set which is compared to the known trajectory of the null distribution. Confidence ellipses can be determined at points along the null trajectory to indicate the consistency of the data with the null distribution.

The second step of the algorithm compares the endpoint of the data trajectory against a chart of the trajectory endpoints for known distributions. The known distribution with an endpoint closest to the data endpoint is used to approximate the pdf of the data. Experiments run at Syracuse University have verified that this procedure requires about 100 data points to yield very good approximations to the underlying distributions.

Two major advantages of this algorithm are recognized. Typical goodness of fit tests accept or reject the hypothesis that the data is from a specified null distribution, but if the hypothesis is rejected they are unable to suggest a distribution which the data most closely approximates. The Ozturk algorithm provides this capability. Second, typical pdf approximation algorithms based on histograms or other methods, [7, 8], may require on the order of thousands of points to obtain a good approximation, while the Ozturk algorithm needs only 100 points.

The small data set required for accurate pdf approximation by the Ozturk algorithm has the potential to make it very effective in nonstationary interference environments, which can be a problem for conventional techniques. Furthermore, the algorithm can be used to approximate multivariate SIRV density functions based on the reduction to a univariate pdf approximation problem as discussed in Section 4.2.6.

4.7 Conclusions

For several reasons related to the characteristics of non-Gaussian, man-made interference and the nature of the M-ary decision problem, it is concluded that spherically invariant random processes are not appropriate models to use for the interference in spread spectrum communication systems. Interference consisting of the sum of multiple, independent, constant envelope jammer waveforms is of particular interest in these systems. The joint probability density function of the samples of this interference can be nonunimodal (more than one local maximum) and have a bivariate marginal density which is not radially symmetric. Neither of these characteristics are satisfied by the class of SIRPs.

Furthermore, non-Gaussian SIRPs do not have an additive closure property. This means that interference which is the sum of independent, non-Gaussian SIRPs does not result in another SIRP, except under very restrictive conditions on the correlation properties of each source.

The minimum probability of error receiver for the M-ary decision problem, where the hypotheses are assumed equally likely, reduces to the linear, Gaussian receiver for all types of SIRV interference when the signal has an unknown amplitude scaling. This result holds whether the unknown amplitude is a random variable or a constant. This means that performance already being obtained by the Gaussian receiver in SIRP interference environments is optimal for this case.

When an unknown random phase is present in the signal, the minimum probability of error receiver does not reduce to the Gaussian receiver. This could lead to useful results for operation in some types of fading channels, where both random amplitude and phase are introduced into the signal by the channel.

The Ozturk algorithm for pdf approximation in the weak signal case has high potential for use in any non-Gaussian interference environment, not necessarily an SIRP.

Bibliography

[1] Muralidhar Rangaswamy, *Spherically Invariant Random Processes for Radar Clutter Modeling, Simulation and Distribution*, Ph.D. dissertation, Syracuse University, Syracuse, New York, 1992.

[2] Prakash Chakravarthi, *Subclutter Visibility: The Problem of Weak Signal Detection*, Ph.D. dissertation, Syracuse University, Syracuse, New York, 1993.

[3] Rajiv Shah, *A New Technique for Distribution Approximation of Random Data*, MS thesis, Syracuse University, Syracuse, New York, 1993.

[4] M. Rangaswamy, P. Chakravarthi, D. Weiner, A. Ozturk, H. Wang, and L. Cai, *Signal Detection in Correlated Gaussian and Non-Gaussian Radar Clutter*, Syracuse University, September, 1992.

[5] H. Van Trees, *Detection, Estimation, and Modulation Theory, Part I*, John Wiley and Sons, New York, 1968.

*[6] M. Desai, J. Dowdle, et al., *Robust Digital Adaptive Transceiver Development Program*, The Charles Stark Draper Laboratory, Inc, RADC-TR-89-332, February, 1990. DOD and DOD contractors only; premature dissemination.

*[7] J. Murphy, P. Tilley, and F. Torre, *Adaptive Nonlinear Coherent Processor Design, Vol. I*, Hazeltine Corp., RADC-TR-89-387 , March, 1990. DOD and DOD contractors only; premature dissemination.

*[8] J. Murphy, P. Tilley, and F. Torre, *Adaptiove Nonlinear Coherent Processor Design, Vol. II*, Hazeltine Corp., RADC-TR-89-387, March, 1990. DOD and DOD contractors only; premature dissemination.

*[9] S. Tyler and J. Patti, *Adaptive Nonlinear Processing for Interference Mitigation in a Spread Spectrum System*, Rome Laboratory, RL-TR-91-248, July, 1991. DOD and DOD contractors only; critical technology.

[10] P. Van Broekhoven, *Digital Density Detector Development Program*, CSDL-R-1787, The Charles Stark Draper Laboratory, Inc., August, 1985.

[11] J. Proakis, *Digital Communications*, 2nd edition, McGraw-Hill Book Co., New York, 1989.

*Although this report references the limited documents noted above, no limited information has been extracted.

[12] F. Oberhettinger and L. Badii, *Tables of Laplace Transforms*, Springer-Verlag, New York, 1973.

[13] S. Kassam, *Signal Detection in Non-Gaussian Noise*, Springer-Verlag, New York, 1988.

[14] K. Yao, "A Representation Theorem and Its Applications to Spherically-Invariant Random Processes," IEEE Transactions on Information Theory, vol. IT-19, pp 600-608, September, 1973.

[15] D. Kelker, "Distribution Theory of Spherical Distributions and a Location-Scale Parameter Generalization," Sankhyā, The Indian Journal of Statistics, Series A, vol. 32, pp 419-430, 1970.

[16] K. Chu, "Estimation and Decision for Linear Systems with Elliptical Random Processes," IEEE Transactions on Automatic Control, vol. AC-18, pp 499-505, October, 1973.

[17] J. Goldman,"Statistical Properties of a Sum of Sinusoids and Gaussian Noise and its Generalization to Higher Dimensions," The Bell Systems Technical Journal, vol. 53, pp 557-580, April, 1974.

[18] J. Goldman,"Detection in the Presence of Spherically Symmetric Random Vectors," IEEE Transactions on Information Theory, vol. IT-22, pp 52-59, January, 1976.

Chapter 5

Radar Weak-Signal Detection Problem

This section discusses the development of radar knowledge appropriate for use in an IPUS-based radar interpretation system for use in high-clutter situations. Applications involving airborne and high-sited ground surveillance radars require the detection of targets in severe ground clutter environments. Our objective in studying this problem was to improve upon conventional radar signal processing techniques, under a specified false alarm probability constraint, by using the signal re-processing IPUS framework to increase the detection probability of weak desired signals in the presence of strong ground clutter. The results of the research are presented in three Ph.D. dissertations and one M.S. thesis, completed under this effort, which appear as appendices in the final report.

In order to develop an optimal signal processing scheme, it is necessary to provide a probabilistic description of the clutter. Previous attempts at clutter modeling have used marginal probability density functions to characterize the clutter. However, radar receivers process N samples in each beam dwell. Consequently, an N-dimensional joint probability density function for the sampled data is needed. Because the clutter is likely to be highly correlated, the samples are not statistically independent. Therefore, the joint probability density function cannot be obtained by simply multiplying together the N corresponding individual marginal densities.

In general, closed form analytical expressions for the joint probability density function of a set of N correlated, non-Gaussian random variables are not unique. In the Ph.D. dissertation by Muralidhar Rangaswamy (see Appendix xxx) entitled, "Spherically Invariant Random Processes for Radar Clutter Modeling, Simulation, and Distribution Identification," it is shown that the spherically invariant random vector (SIRV) provides an elegant and mathematically tractable technique for characterizing the probability density function of a correlated, non-Gaussian clutter vector. Although the SIRV family does not encompass all random processes, it goes well beyond the Gaussian family. The following is a partial list of probability distributions that fall within the SIRV family: 1) Chi, 2) Weibull, 3) Generalized Rayleigh, 4) Rician, 5) Laplace, 6) Cauchy, 7) K-distributed, 8) Student-t, and 9) Gaussian.

A brief review of spherically invariant random processes is provided by Dennis Stadelman in Section xxx.2. A more detailed discussion is given in Rangaswamy's Ph.D. dissertation (see Appendix E). Chapter 1 provides an introduction to the fundamental issues addressed in the dissertation, which are:

1. specification of suitable statistical models for radar clutter,
2. development of efficient computer simulation procedures for generating samples characterized by the various statistical models,
3. development of an identification procedure for fitting one or more statistical models to a set of experimental data.

In Chapter 2 a review of the literature, as it pertains to the modeling of radar clutter as a spherically invariant random process, is presented. Chapter 3 develops techniques for obtaining the joint probability density function of N complex, non-Gaussian random variables, assuming that the clutter can be characterized as a spherically invariant random process. The need for a library of multivariate, non-Gaussian probability density functions is discussed. Several examples illustrating the various techniques for specifying the mulitivariate, non-Gaussian probability density function are provided. Finally, a key result for identifying the multivariate, non-Gaussian probability density functions arising from spherically invariant random processes is presented. Chapter 4 deals with the problem of computer generation of correlated, non-Gaussian radar clutter that can be characterized as a spherically invariant random process. Two canonical simulation procedures are presented. A graphical goodness-of-fit test is introduced to validate the simulation procedures. Chapter 5 is concerned with the distribution identification of radar clutter characterized by spherically invariant random processes. A new graphical scheme based on a key result presented in Chapter 3 is used to address the distribution identification problem. This procedure reduces the multivariate distribution problem to an equivalent univariate distribution identification problem, resulting in considerable computational simplicity. Finally, a new technique for shape parameter estimation is suggested based on the identification procedure. The chief advantage of this scheme is that relatively few samples are needed for the distribution identification problem. Chapter 6 concludes with a summary and suggestions for future research.

Having developed models for correlated, non-Gaussian radar clutter samples, attention was then focused on the weak signal detection problem in a strong non-Gaussian clutter background. The concept of the locally optimum detector (LOD) was used to address this problem. A brief review of LODs is given by Dennis Stadelman in Section xxx.3. A more detailed discussion is provided in the Ph.D.

dissertation by Prakash Chakravarthi (see Appendix F) which is entitled, "Subclutter Visibility: The Problem of Weak Signal Detection."

Chapter 1 of Chakravarthi's Ph.D. dissertation discusses the key elements of the weak signal detection problem. A literature review on weak signal detection and derivations of the LOD are presented in Chapter 2. It is shown that the LOD determines whether a target is present or absent by comparing a statistic computed from the data to a set threshold. Both deterministic and random target signal are considered. The receiver structures are specialized to the case for which the clutter plus noise can be approximated by a spherically invariant random process. Since the clutter is assumed to be non-Gaussian, the LOD receiver structure turns out to be nonlinear. As a result, system performance must be determined by means of computer simulation. The threshold is conventionally determined through a Monte Carlo procedure. Unfortunately, the number of trials is inversely proportional to the false alarm probability, P_F. For example, when $P_F = 10^{-6}$, a minimum of ten million trials needs to be generated.

To avoid carrying out so many trials, a new technique, based on extreme value theory, is presented in Chapter 3. It is demonstrated that fairly accurate thresholds can be determined for false alarm probabilities as small as 10^{-7} with as few as 10,000-30,000 trials. Assuming that the clutter plus noise can be approximated by either the multivariate Student-t or K-distributions, the LOD is developed in Chapter 4 for the weak signal detection problem. The system performance is evaluated by means of computer simulation for each distribution. It is shown that the performance improvement for the LOD is significant compared to the Gaussian receiver. A new technique called the amplitude dependent locally optimum detector is developed in Chapter 5. It is demonstrated that significant performance improvement can be obtained compared to the LOD when the clutter plus noise is multivariate K-distributed. A summary and suggestions for future research are presented in Chapter 6.

The type of clutter present in a received set of N samples will not be known in advance. Consequently, it is necessary to monitor the environment and identify the underlying probabilistic clutter model. This falls into the category of a model variety problem since experimental clutter data, collected at different times and under different conditions, has been shown to fit a variety of probability distributions. A new algorithm, called the Ozturk algorithm, was developed under this effort for the purpose of analyzing and characterizing random data. The algorithm has two modes of operation. In the first mode, a goodnes-of-fit test is performed. This mode allows for determination of whether a given set of data is statistically consistent with a prespecified probability density function. In the second mode, an approximation procedure is carried out. In particular, given the data, the algorithm determines a probability density function that gives a good approximation to the probability distribution from which the data was generated. The Ozturk algorithm works well in both modes of operation, even when presented with as few as 100 data samples.

A brief review of the Ozturk algorithm is given by Dennis Stadelman in Chapter 4 of this report. A more detailed discussion is provided in the Masters thesis by Rajiv Shah (see Appendix D) which is entitled, "A New Technique for Distribution Approximation of Random Data." Advantages and disadvantages of well-known goodness-of-fit tests are discussed in Chapter 1 of the thesis. The Ozturk algorithm is developed in Chapter 2. Computer simulation results using the Ozturk algorithm are presented in Chapter 3. Chapter 4 concludes the thesis with a summary and suggestions for future work.

The data in a radar surveillance volume is likely to be nonhomogenous and nonstationary. The Ph.D. dissertation by Mohamed Slamani (see Appendix G of this report) which is entitled, "A New Approach to Radar Detection Based on the Partitioning and Statistical Characterization of the Surveillance Volume," explores use of an expert system based on the IPUS philosophy, to assist with monitoring of the environment. Chapter 1 discusses some of the difficulties that arise in the classical radar detection problem. Their solution is proposed in Chapter 2 which uses an expert system with feed-forward processing. In Chapter 3 an improved solution is presented using feed-back processing. The general radar detection problem is described in Chapter 4 and a mapping procedure is introduced to separate between background noise and clutter patches. In Chapter 5 an image processing technique is developed for the mapping procedure. Next, an indexing procedure is developed in Chapter 6 to enable the investigation of clutter subpatches and the approximation of probability distributions for each clutter patch. Finally, expert system rules are developed in Chapter 7 to enable the expert system to control both the mapping and indexing stages. A summary and suggestions for future research are given in Chapter 8.

The major significant results achieved during this research effort are listed below:

1. The capability for modeling correlated, non-Gaussian radar clutter was greatly enhanced by the creation of a large library of multivariate probability density functions based upon the concept of spherically invariant random processes.

2. An ability to perform computer simulations involving correlated, non-Gaussian radar clutter was made possible by development of two canonical schemes for generating the clutter.

3. The ability to completely characterize univariate random data by processing a relatively small number of samples was made possible by development of the Ozturk algorithm.

4. A theoretical procedure was devised for reducing the multivariate probability distribution approximation problem dealing with spherically invariant random processes to an equivalent univariate distribution approximation problem where the Ozturk algorithm is applicable.

5. By developing weak signal detectors for correlated, Student-t and K-distributed clutter environments and evalutating their performance, it was demonstrated that non-Gaussian receivers designed to optimize performance in non-Gaussian environments can significantly outperform the conventional Gaussian-based receivers.

6. A reduction by several orders of magnitude in the number of trials needed for Monte Carlo simulations of radar receivers was made possible by development of a new technique for setting the receiver threshold based upon extreme value theory.

7. Mapping and indexing procedures were developed and demonstrated via several examples for adaptively partitioning a radar surveillance volume into clutter and background noise regions and for approximating the underlying probability distribution of each region.

8. Expert system rules were developed for enabling the expert system based on the IPUS philosophy to control both the mapping and indexing stages during the adaptive partitioning of a radar surveillance volume.

Chapter 6

RESUN Framework Improvements

6.1 Introduction

IPUS is built on the RESUN sensor interpretation framework. As a result, development of the generic IPUS testbed has involved a significant amount of work refining or generalizing elements of the RESUN model, and extending the existing RESUN system implementation. For example, in scaling up the testbed to handle more source models and more complex sources, we had to make use of more top-down, expectation-driven strategies to control the number of source models that would be considered. These strategies are based on information from approximate processing methods—e.g., spectral bands which are based on energy present at specific frequency bands. This caused us to have to extend RESUN's evidential representation and evidence summarization process. While changes such as this were easily accommodated within the RESUN framework, they did require a significant amount of effort to properly support.

Among the key refinements of RESUN for IPUS are:

- canonical evidence combination methods for evidence summarization process;
- representing negative evidence uncertainty in the evidence summaries;
- representation and summarization of approximate knowledge;
- additional control plan sequencing constructs;
- event-based refocusing;
- improved RESUN implementation.

6.2 A Brief Review of the RESUN Framework

One of the key motivations for the development of RESUN was to extend the set of methods that could be used for interpreting sensor data. In complex domains involving noise and masking, no single method is sufficient. For example, an interpretation system must be able to use differential diagnosis as well as hypothesize and test methods, and specialized knowledge for particular situations. RESUN provides a mechanism to apply sophisticated control strategies, using large amounts of context-specific knowledge to carefully control the search for most-likely interpretation. The Key Components of the RESUN framework are:

- Symbolic source of uncertainty statements (SOUs) that represent the reasons hypotheses are uncertain and drive identification of actions to resolve uncertainty (with interpretation viewed as abductive inference);
- A high-level model of composite interpretation uncertainty relative to the termination criteria (system goals);
- A script-based planner for control, which provides context-specific knowledge and reasoning, maintains the opportunism of blackboard-based systems, and can implement method/decision search.

In the RESUN control planning framework, interpretation strategies are encoded as a set of control plan schemes and plan-specific focusing heuristics. Each plan specifies a sequence of subgoals that must be accomplished to meet the goal of the plan. The focusing heuristics control the instantiation of each plan (based on what is considered most appropriate for the current situation) by selecting the plans that are used to pursue the subgoals and by selecting the objects to be used to instantiate subgoals (i.e., selecting among possible alternative plan-variable bindings). A plan subgoal sequence is encoded using a shuffle grammar, which includes a set of operators that specifies the temporal relations among the subgoals. For example, the :SEQUENCE operator includes a fixed list of subgoals, each of which is pursued following the successful completion of the previous subgoal.

One of the key aspects of the RESUN interpretation framework is the use of symbolic source of uncertainty statements (SOUs). The SOUs provide more information than numeric belief ratings. They allow the system to understand the reasons why its current evidence is uncertain so that it can identify appropriate methods to resolve this uncertainty. However, the system still needs numeric evaluations of the degree of belief in the hypotheses and the effects of the various SOUs in order to evaluate the termination criteria and to reason about control decisions. For example, in deciding which hypotheses to work on next, the system usually must consider the belief rating for each hypothesis relative to the termination criteria and to the belief in each of the other hypotheses. It must then usually consider which hypothesis SOUs are most critical to resolve.

```
#<source-ext.00006 burglar-alarm>
  RATING: 0.12307032
  PARTIAL-EVIDENCE-UNCERTAINTY: 0.29800
  POSSIBLE-ALT-EXPLANATION-UNCERTAINTY: 0.74052006
  POSSIBLE-ALT-SUPPORT-UNCERTAINTY: 0.0
  CONSTRAINT-UNCERTAINTY: 0.0
  ALT-EXTENSIONS-UNCERTAINTY:  0.045408897
  TOP-LEVEL-ALT-EXTENSIONS:
    (<ALT-EXT-UNIT Alt:#<source-ext.00004 A>>)
  NEGATIVE-EVIDENCE-UNCERTAINTY: 0.18336463
  NEGATIVE-EVIDENCE-EXPLANATION: 0.09136386
```

Figure 6.2.1: An example evidence summary.

Because of the need to make such evaluations and comparisons, RESUN's evidential representation system includes a framework for numerically summarizing evidence based on the SOUs. However, instead of computing a single-number belief rating, the summarization process produces a composite characterization of the uncertainty in a hypothesis in terms of an overall belief rating and the amount of uncertainty contributed by the different classes of SOUs (associated with the hypothesis). The major elements of the composite rating are: possible alternative explanation uncertainty, possible alternative support uncertainty, constraint uncertainty, negative evidence uncertainty, and alternative extension uncertainty. Thus, for any belief rating b, where b ¡ 1, the sum of the ratings for these uncertainty classes would add up to 1-b (the situation for the negative evidence is a bit more complicated actually). In addition to these categories, the composite includes a partial support uncertainty rating that indicates how much of the remaining uncertainty might potentially be reduced by gathering additional evidence for the hypothesis.

These SOU-class ratings summarize the SOUs by giving an abstract indication of the reasons why the hypothesis is uncertain (i.e., not fully believed). Having the composite rating allows for more detailed reasoning than would be possible with a single number rating. For example, it can distinguish between a hypothesis that has low belief due to a lack of evidence and one for which there is negative evidence. This capability is used in distinguishing between potential answer hypotheses that should be modeled as "answers" and those that should be modeled as "non-answers." Potential answer hypotheses may not be currently believed (i.e., belief rating ¡ 0.5 more likely wrong than right) simply because not enough evidence has been gathered to resolve the inherent abductive uncertainty (resulting from alternative possible explanations for the supporting data). These hypotheses are modeled (in PS-Hyp) as potential answers so that the system will attempt to prove them correct. On the other hand, potential answer hypotheses may not be believed because more evidence has been gathered against them than for them. Such hypotheses are modeled as non-answers so that the system will attempt to disprove them. In either case, of course, the system may pursue the hypotheses until sufficient evidence is gathered to reach belief levels specified in the system goals.

The summarization process operates by recursively summarizing the support evidence for a given top-level hypothesis. Summarization is carried out using application-specific evaluation functions because neither Bayes' Rule nor Dempster-Shafer's Rule are generally applicable to interpretation due to the lack of independence of hypothesis evidence. Nonetheless, the application-specific evaluation functions effectively compute conditional probabilities and the composite rating permits these evaluation functions to be quite modular. Briefly, the summarization process for a hypothesis is:

1. Evaluate the support evidence and produce composite ratings that are attached to each corresponding uncertain-support SOU in the hypothesis;
2. Evaluate all the negative evidence producing ratings for the negative evidence SOUs;
3. Combine the support evidence to get a support rating for the hypothesis;
4. Reduce the belief based on the possible-alternative-explanation SOUs associated with the hypothesis extension's explanation;
5. Reduce the belief further based on the uncertain constraint SOUs associated with the hypothesis extension's explanationthis gives the support rating that this hypothesis provides to its explanation;
6. Stop at the top-level hypothesis (no explanation);
7. Identify and compare alternative top-level hypotheses (denoted by alt-extension SOUs) and reduce the hypothesis belief ratings to account for the alternatives.

6.3 Canonical Evidence Combination Methods

RESUN's evidence summarization process requires that there be appropriate *evidence combination methods*: functions for combining the evidence summaries that will occur with each type of hypothesis to produce a belief summary for a hypothesis. Combination functions are a key issue in developing a RESUN application because they may be specific to each hypothesis (abstraction) type and they may be very complex. Thus, one of the major areas of effort in implementing a RESUN application is to provide numeric summarization methods for each hypothesis type. One of the goals of IPUS research has been to investigate general methods for combining/summarizing the SOUs that are appropriate for the characteristics of the signal understanding problem. Major effort has been devoted to developing an appropriate set of canonical models for evidence combination—e.g., table driven for discrete evidence and functions for continuous support evidence.

The current IPUS applications have provided us with characteristics that were not part of previous RESUN applications:

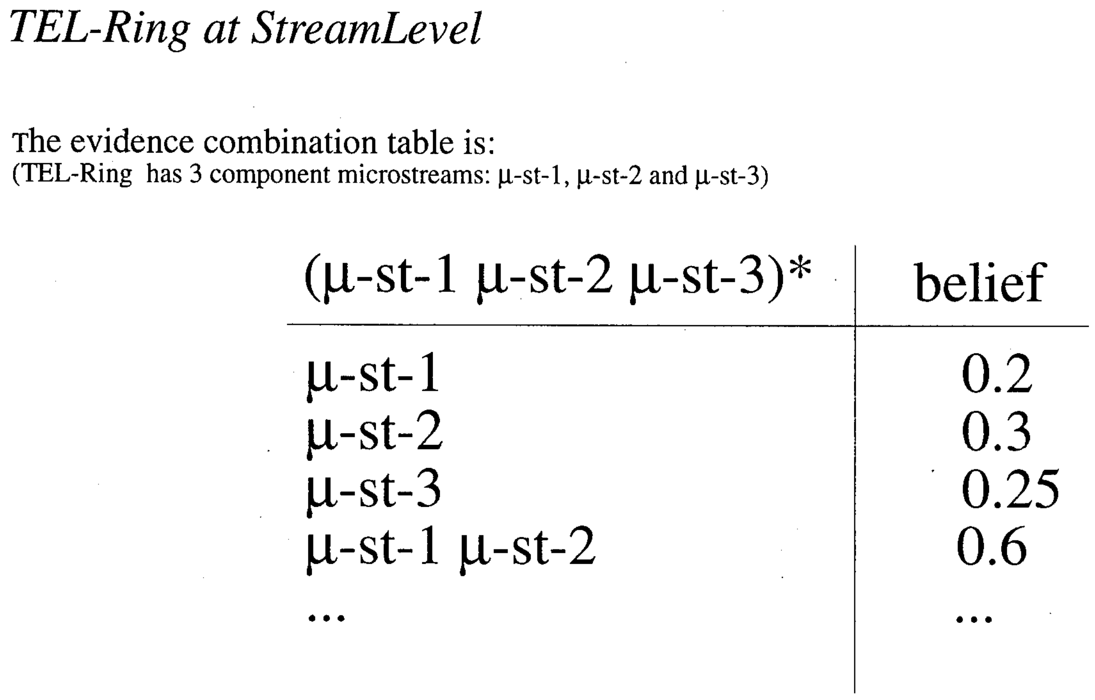
- Non-additive evidence combination ($P(A|B,C) >> P(A|B) + P(A|C)$). For example, consider a telephone ring with three frequency components a, b and c. Component a in isolation would give a belief of .2 to the source. Component b in isolation would give a belief of .3. Components a and b together should give more than a belief of .5 (.2 + .3) to the source, to reflect the fact that one frequency component alone could easily belong to another source, but when two of them are found the chances of them belonging to other sources are smaller.

- Highly non-uniform support—some pieces of evidence are much more critical than others.

- Asymmetric effect of positive and negative evidence—e.g., finding a support component or not knowing about the existence that component can have a relatively weak effect on belief, but failing to find it can have a strong effect. This effect is most obvious in the verification of a source's components. For example, consider a source that should always have a very low energy component at a certain frequency. The presence of this component would not contribute much to the belief in the source, but if looked for and not found it would strongly decrease the belief in the source.

- The evidence hierarchy contains both discrete and (quasi) continuous types of support evidence. In the acoustic domain, a stream is formed by a discrete set of microstreams, while a microstream is a continuous type of event—i.e., a sequence of contours in time. The belief in a microstream is computed based on the length of its support, independently of how many or which contours support it. The belief in a stream is computed based on how many and which components support it.

These characteristics have led us to develop a language for representing source specific evidence combination methods. To support the previous characteristics, we developed abstraction-specific evidence combination methods. The evidence combination method at the source abstraction level must be capable of combining periodic, simultaneous and sequential events. In our approach, a different method exists for



TEL-Ring at StreamLevel

The evidence combination table is:
(TEL-Ring has 3 component microstreams: μ-st-1, μ-st-2 and μ-st-3)

(μ-st-1 μ-st-2 μ-st-3)*	belief
μ-st-1	0.2
μ-st-2	0.3
μ-st-3	0.25
μ-st-1 μ-st-2	0.6
...	...

Figure 6.3.2: An example evidence combination table.

each combination class: 1) periodic: step function, 2) sequential: table indexed by elements in the sequential events' power set, and 3) simultaneous: table indexed by elements in the simultaneous events' power set. In a similar way, the evidence combination method at the next lower level, the stream level, must be capable of reflecting the non-additive nature of evidence and the asymmetric effect of positive and negative evidence. The combination method chosen is a table indexed by the elements in the components power set.

The evidence combination method at this level must be capable of reflecting the continuous nature of the source. The combination method chosen is an S-shape function of the form $g(x) = 1 - e^{-f(x)}$, where $f(x)$ is a monotonically increasing function. $g(x)$ is shaped for each component by providing three values a, b, c; where a is the length of support necessary for the component to have a belief of 0.5, b is the length of support necessary for the component to have 0.9 belief, and c is the length of support necessary for the component to have 1.0 belief.

6.4 Negative Evidence Uncertainty

Another RESUN evidential issue that has been investigated as part of the IPUS project has been the handling of uncertainty in negative evidence. Just as with positive (confirming) evidence, negative (disconfirming) evidence is uncertain because

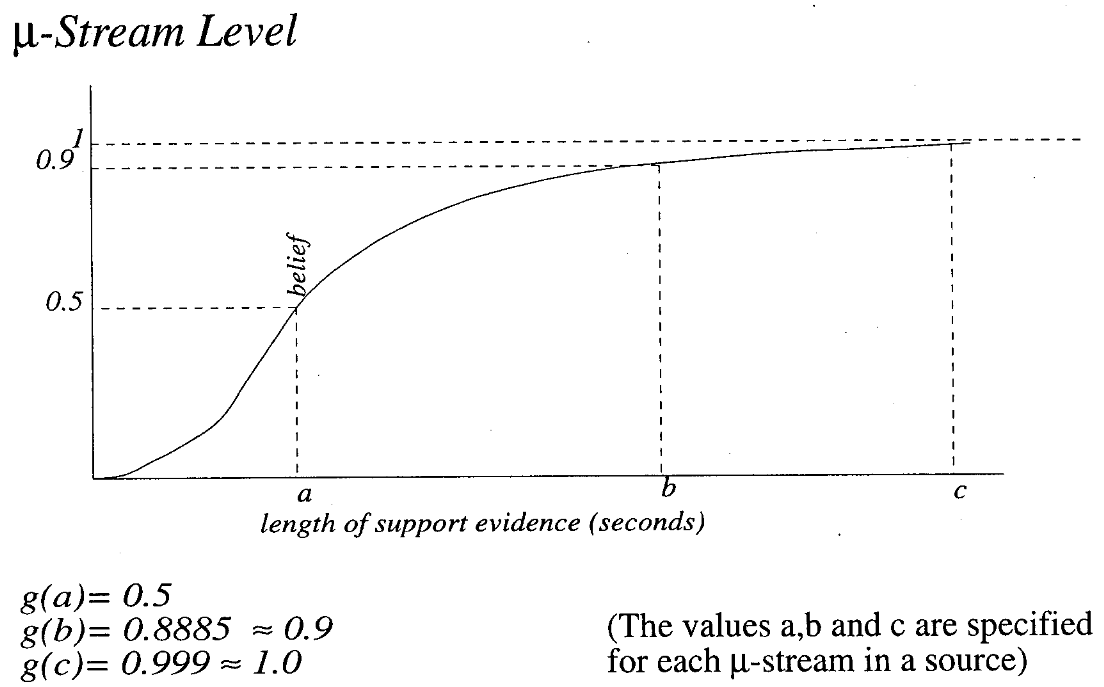


Figure 6.3.3: An example of an evidence combination function.

there may be explanations for the failure to find the needed supporting data or hypotheses. Negative evidence uncertainty was always part of the RESUN model, but it had not been extensively explored with earlier RESUN applications because there were limited sources of uncertainty for negative evidence. In IPUS, this is a key issue since it is related to the notion of discrepancies and reprocessing of data. For example, failure to find a frequency component suggests that the source hypothesis is incorrect—i.e., the failure creates negative evidence for the source. However, the inability to find the desired component may be due to an inappropriate SPA having been used. As a result, this negative evidence will not be certain/conclusive—it will be uncertain.

In RESUN, negative evidence uncertainty is represented by "negative evidence explanation" SOUs that are associated with negative evidential inferences. These SOUs represent the probability of there being explanations for a failure to find a support—i.e., they represent the level of belief that the negative evidence is actually disconfirmatory, as opposed to it simply being the result of incorrect processing, etc. During evidence summarization, negative evidence explanation SOUs lead to a belief range for affected hypotheses: the belief given that the negative evidence is correct vs. given the probability that it is incorrect

The beliefs/probabilities for negative evidence explanation SOUs are computed on the basis of domain knowledge. For example, assume that expected support (peaks) for a frequency track is not found. If the energy-threshold parameter for

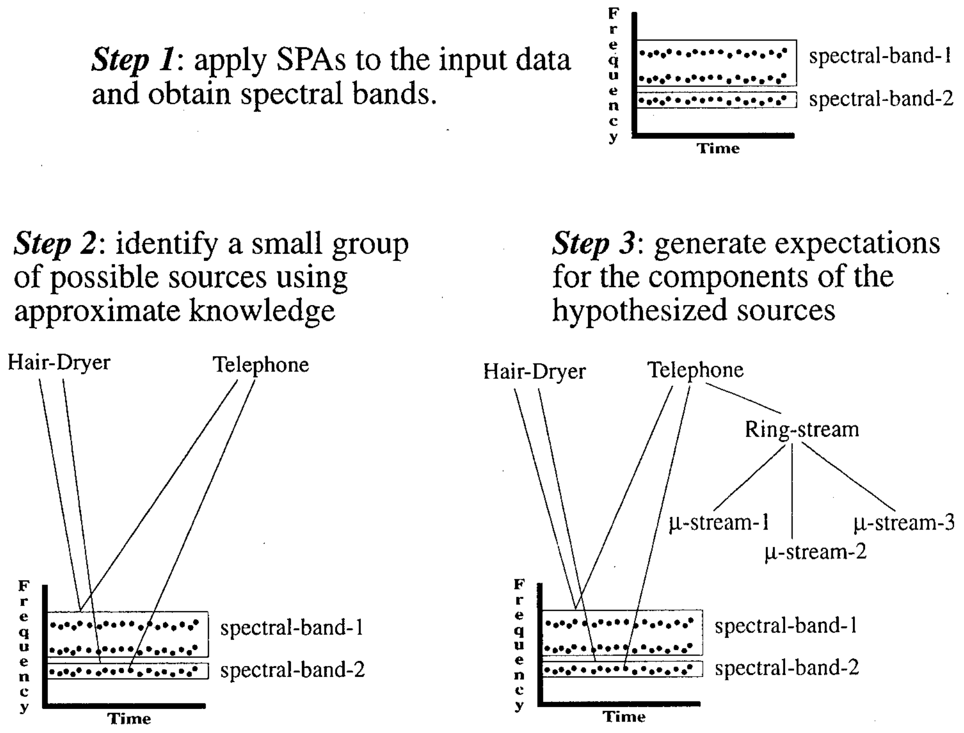

Step 1: apply SPAs to the input data and obtain spectral bands.
Frequency
Time
spectral-band-1
spectral-band-2
Step 2: identify a small group of possible sources using approximate knowledge
Hair-Dryer
Telephone
Step 3: generate expectations for the components of the hypothesized sources
Ring-stream
μ-stream-1
μ-stream-2
μ-stream-3

Figure 6.5.4: An example of spectral bands approximate knowledge.

the peak-picking algorithm was higher than the energy expected for the component, clearly the negative evidence against the track must be rated as highly uncertain and only moderately disconfirmatory. One of the actions that a system can take to resolve uncertainty is to simply gather evidence to refine the probability assessment in a negative evidence explanation SOU.

6.5 Approximate Knowledge in IPUS

One of the major changes in the control strategies during the IPUS project has been the introduction of approximate processing, using spectral bands. Approximate knowledge can be used to reduce the cost of interpretation search by using inexpensive evidence to focus the search before using more expensive "exact" evidence. In IPUS, the approximate processing strategies introduce a stronger top-down direction. With approximate processing, we first use (inexpensive) spectral bands knowledge to identify a small set of possible sources and then refine this set by generating support component expectations for the sources, performing focused/top-down contouring, and verifying the component expectations.

In RESUN, the uncertainty associated with approximate knowledge is represented by SOUs that are associated with the evidential inferences, which reflect the constraints that are unchecked as a result of the evidence being approximate. Use

Step 4: do contouring for the telephone hypothesis, or reprocess the SPAs if the parameters used previously are not appropiate for the telephone hypothesis

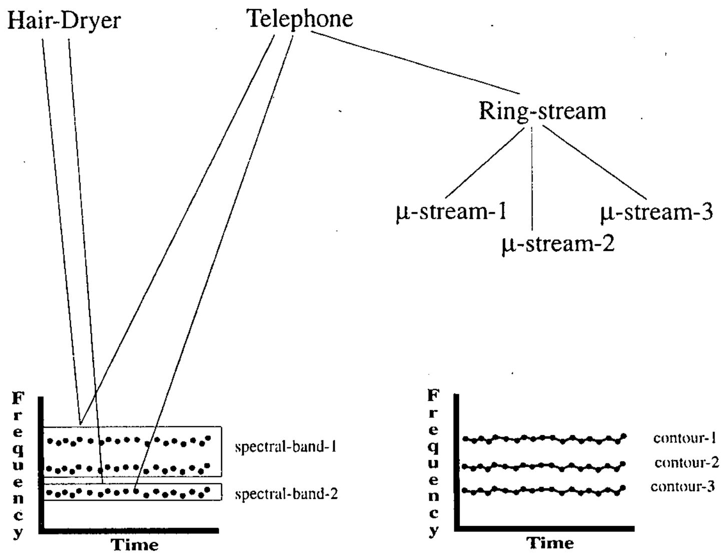


Figure 6.5.5: Spectral bands example, continued.

Step 5: verify the telephone microstreams. Whenever a contour supports a microstream, use this new information to refine the spectral band.

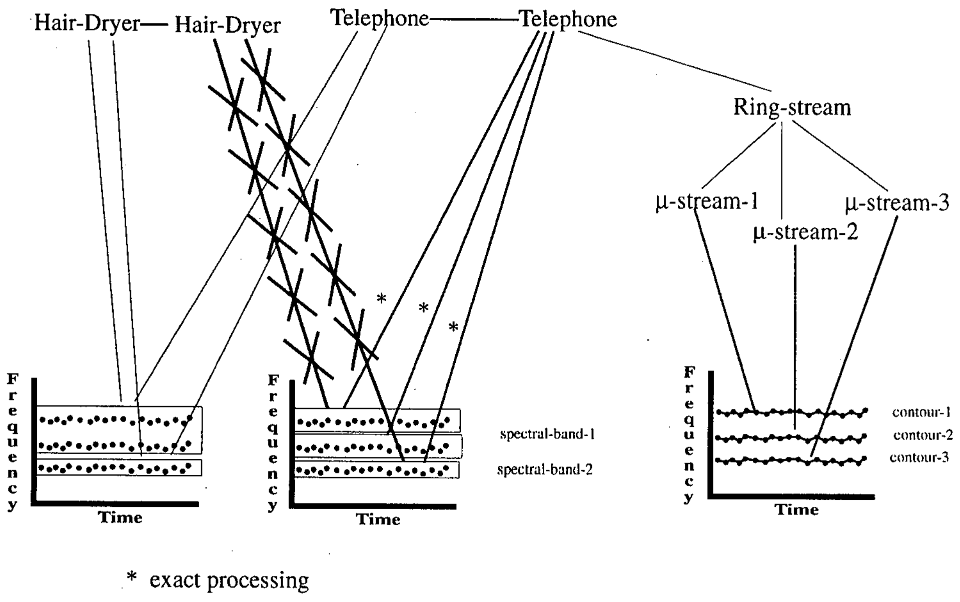


Figure 6.5.6: Spectral bands example, continued.

of approximate knowledge requires methods for: combining approximate and exact sources of evidence; refining the approximate evidence as the result of gathering exact evidence for some of the explanations; and propagating belief among the multiple explanations of the approximate evidence (which may be consistent or may be alternatives).

In combinding approximate and exact knowledge, evidence from approximate knowledge is considered as evidence from a different source of evidence. We have different evidence combination methods for each source of evidence. In order to compute the belief in a hypothesis with evidence from multiple sources of evidence, these must be combined together. Exact processing is preferred over approximate processing when computing the belief on a hypothesis. That is, if a source component has evidence from approximate processing and from exact processing, the exact processing evidence will be used. The approximate knowledge provides less belief than the exact knowledge due to uncertainty reflecting the uncheckable constraints.

6.6 Additional Control Plan Sequencing Constructs

The initial set of RESUN control plan grammar operators (for sequencing subgoals) was never intended to be exhaustive. As part of the IPUS research we have been looking for additional operators that are necessary or useful for interpretation applications. A few new operators were added as a result of extending the strategies in the IPUS testbed, and several more have been contemplated for future addition.

One of the added operators is :LIST-SHUFFLE. This operator is a parallel subgoal construct like :SHUFFLE, but unlike :SHUFFLE it does not specify a fixed set of parallel subgoals. Instead, it specifies a single subgoal type, whose number of parallel instantiations will be determined dynamically from a (list-valued) plan variable. While this operator has no effect on performance given that we do not yet have a parallel processor implementation, it can make it easier and clearer to write plans that formerly would have had to use :ITERATION to accomplish a list of subgoals.

Another new grammar operator is the :ITERATION-OR operator. This operator is derived from the :ITERATION operator, which specifies that an indeterminate number of instances of a subgoal need to be iteratively pursued. However, whereas the :ITERATION operator requires that all subgoal instances have been successfully pursued for the plan to be considered successful, the :ITERATION-OR operator requires only that at least one of the subgoal instances have been successful. Clearly, "or versions" of other operators might also be needed—e.g., :SEQUENCE-OR, :LIST-SHUFFLE-OR.

6.7 Event-Based Refocusing

The refocusing mechanism is one of the key innovations of the RESUN control planner. It makes it possible to reconsider planner focusing decisions based on the evolving characteristics of the problem-solving situation. This allows for opportunistic control and a search process for making control decisions. For example, current methods for pursuing a source hypothesis might be interrupted if new data becomes available or an alternative source might be pursued if little progress is being made on the previously selected source hypothesis. When a decision point is specified as a refocusing point, the associated focusing heuristic provides the conditions under which the decision should be reconsidered.

One of the key issues in implementing refocusing is determining when conditions should be checked during the control planning cycle. In order to make this process as efficient as possible, refocusing conditions are now classified by the types of event in which they are interested (only certain kinds of changes can have occurred at particular points in the planning cycle). For example, :ACTION refocusing conditions are interested only in changes to the interpretation hypotheses that can occur only after an action is executed, while :UPDATE refocusing conditions are interested in changes in the status of control plans or subgoals.

Another advantage of this classification is that the refocusing conditions are now passed information about the particular event that triggered their checking. This can drastically reduce the need to search the planning structure or the database to see if relevant changes have occured—since it can often be immediately determined whether the event might have affected the status of the condition or not.

6.7.1 Monitoring Subgoals

Subgoals can now be set up as "monitoring subgoals" to monitor for periodic events. This allows the user to write conjunctive goals in which one of the subgoals is something like, "and handle any x events that might occur while accomplishing the other subgoals." Thus, monitoring subgoals are intended to be used in a :SHUFFLE construct in parallel with some other non-monitoring (active) goals. Unlike non-monitoring subgoals, when a monitoring subgoal is placed infocus it is not immediately pursued to find matching plans, etc. Instead, the associated monitoring condition function is placed onto a list of monitoring conditions to be checked, and the subgoal will be activated and pursued only when/if its condition is satisfied. Should all of the non-monitoring subgoals of a :SHUFFLE construct complete while one or more monitoring subgoals have yet to be activated, the monitoring subgoals are considered to have completed satisfactorally. Previously, uncertain periodic events had to handled by plans that would periodically actively look to see if the events had occured (i.e., polling vs. interrupt-driven).

6.8 RESUN Implementation Improvements

We have also made a number of changes to the RESUN system implementation that facilitate control plan development and improve the efficiency of the system, as well as more fully completing some aspects of the original implementation:

- subgoal focusing and refocusing (for the parallel subgoal constructs) have been fully implemented;
- refocusing now has complete freedom to repeatedly remove from focus and then return to focus, focusing options for any of the types of refocusing points;
- improved the control plans syntax to make control plans both shorter and simpler, which should reduce the chance of some coding errors;
- extended the plan verification routines so that more control plan errors can be identified at load time versus run time;

- expanded the tracing options to improve debugging by providing additional information options as well as the option to print information for select plan or subgoal types to prevent information overload;
- the plan structures were simplified so that they are easier to examine and use less storage;
- reduced the use of (dynamic) list data structures so that less garbage collection is required during long system runs.

6.9 Conclusions

One of the goals of our research has been to evaluate whether the RESUN sensor interpretation architecture is suitable for encoding the complex and dynamic control strategies necessary for signal understanding systems based on the IPUS reprocessing architecture. IPUS has provided many opportunities for exploring aspects of the RESUN architecture that were little used by previous experimental applications. While this required us to spend significant effort refining the RESUN model and extending the RESUN implementation, we did not come across any major problems in handling the issues the have arisen in IPUS.

Appendix A

Publications

1994

Carver, N. and Lesser, V., "The Evolution of Blackboard Control Architectures," *Expert Systems with Applications Special Issue on the Blackboard Paradigm and Its Applications.* Vol. 7, No. 1, Liebowitz (Ed.). New York: Pergamon Press, pp. 1–30, Jan.–Mar. 1994.

Dorken, E. and S. H. Nawab, "Conditioned Constant-Q Spectra for Improved Frequency Tracking in the Presence of Interfering Signals," *Journal of the Acoust. Soc. Am.*, vol. 95, no. 4, pp 2059-67, April 1994.

Dorken, E. and S. H. Nawab, "Frame-Adaptive Techniques for Quality versus Efficiency Tradeoffs in STFT Analysis," *Proceedings of the 1994 IEEE International Conference on Acoustics, Speech and Signal Processing*, Adelaide, Australia, April 1994.

Dorken, E. and S. H. Nawab, "Improved Musical Pitch Tracking using Principal Decomposition Analysis," *Proceedings of the 1994 IEEE International Conference on Acoustics, Speech and Signal Processing*, Adelaide, Australia, April 1994.

Lesser, V.R., Nawab, H. and Klassner, F., "IPUS: An Architecture for the Integrated Processing and Understanding of Signals," accepted for publication in *AI Journal*, 1994.

Nawab, S. H. and Dorken, E. "A Framework for Quality Versus Efficiency Tradeoffs in STFT Analysis," *IEEE Transactions on Signal Processing* (under review).

Paneras, D. E. , R. Mani and S. H. Nawab, "STFT Computation Using Pruned FFT Algorithms," *IEEE Signal Processing Letters*, vol. 1, no. 4, April 1994.

Slamani, M.A., DD. Weiner, and A. Ozturk, "A New Approach to Scene Analysis for IR Images," *Proceedings of the 1994 37th Midwest Symposium on Circuits and Systems.*

1993

Bhandaru, M., Draper, B. A., and Lesser, V., "Learning Image to Symbol Conversion," AAAI 1993 Fall Symposium Series: Machine Learning in Computer Vision: What, Why and How? Raleigh, North Carolina, October 1993.

Chakravarthi, P., Weiner, D. and Slamani, M. "Performance of the Locally Optimum Detector in a Correlated K-Distributed Disturbance," *Proceedings of the 1993 National Radar Conference*, Wakefield, MA, April 20–22, 1993.

Chakravarthi, P., "Subclutter Visibility: The Problem of Weak Signal Detection," Ph.D. Dissertation, Syracuse University, March 1993.

Dorken, E. "Approximate Processing and Knowledge-Based Reprocessing of Non-Stationary Signals," Ph.D. Thesis, Department of Electrical, Computer and Systems Engineering, Boston University, September 1993.

Dorken, E. and Nawab, S.H. "A C++ Shell For Knowledge-Based Radar Signal Processing," presentation at the IEEE Conference on Dual Use Technologies and Applications, State University of New York, Utica/Rome, New York, May 1993.

Dorken, E. and Nawab, S.H. "Time-Frequency Analysis of Non-Stationary Harmonic Sounds," *Proceedings of the 1993 IEEE International Conference on Acoustic, Speech and Signal Processing*, Minneapolis, MN, Vol. 3 pp. 249–52, April 1993.

Klassner, F., Lesser, V., Nawab, H. "Fusing Multiple Reprocessings of Signal Data," SPIE Sensor Fusion VI Conference, *SPIE Proceedings*, Vol. 2059, 1993.

Lesser, V., Nawab, H., Gallastegi, I., Klassner, F., "IPUS: An Architecture for Integrated Signal Processing and Signal Interpretation in Complex Environments," *Proceedings of the Eleventh National Conference on Artificial Intelligence*, 1993.

Nawab, S. H., Beyerbach, D., Dorken, E., "Principal Decomposition of Time- Frequency Distributions," *IEEE Transactions on Signal Processing*, vol. 41, no. 11, pp. 3182-86, Nov. 1993

Nawab, S.H. and Dorken, E. "Efficient STFT Approximation Using a Quantization and Differencing Method," *Proceedings of the 1993 IEEE International Conference on Acoustic, Speech and Signal Processing*, Minneapolis MN, vol. 3, pp. 587–90, April 93.

Slamani, M. and Weiner, D. "Rationale Behind the Use of Image Processing to Partition a Radar Surveillance Volume into Background Noise and Clutter Patches," *Proceedings 36th Midwest Symposium on Circuits and Systems*, Detroit, MI, August 1618, 1993.

Slamani, M. and Weiner, D. "A Knowledge-based AI Approach to Analysis of a Radar Surveillance Volume," an oral presentation at the 1993 Dual-Use Technologies and Applications Conference, Utica, NY, May 24–27, 1993.

Slamani, M. and Weiner, D. "Use of Image Processing to Partition a Radar Surveillance Volume into Background Noise and Clutter Patches," *Proceedings of the 1993 Conference on Information Sciences and Systems*, Johns Hopkins University, Baltimore, MD, March 24–26, 1993.

<u>1992</u>

Bitar, N., Nawab, S.H., Dorken E., Paneras, D. E., "Integration of STFT and Wigner Analysis in a Knowledge-Based System for Sound Understanding," *Proceedings of the 1992 IEEE International Conference on Acoustic, Speech and Signal Processing*, San Francisco, CA, vol. 4, pp. 585-88, March 1992.

Chakravarthi, P., Weiner, D., Ozturk, A., "Performance of the Locally Optimum Detector in a Correlated Non-Gaussian Disturbance," 35th Midwest Symposium on Circuits and Systems, August 9-12, 1992, Washington, DC.

Dorken, E., Nawab, S.H., Lesser, V., "Extended Model Variety Analysis for Integrated Processing and Understanding of Signals," *Proceedings of the 1992 IEEE International Conference on Acoustic, Speech and Signal Processing*, San Francisco, CA, vol. 5, pp. 73-76, March 1992.

Klassner, F., "Data Reprocessing and Assumption Representation in Signal Understanding Systems," TR. 92-52 Computer Science Dept., University of Massachusetts, Amherst, MA.

Rangaswamy, M., "Spherically Invariant Random Processes for Radar Clutter Modeling, Simulation, and Distribution Identification," Ph.D. Dissertation, Syracuse University, December 1992.

Slamani, M. and Weiner, D., "Comparative Study of the Transversal Filter with Conventional DFT Processing in a Nonlinear Radar Receiver," 35th Midwest Symposium on Circuits and Systems, August 9-12, 1992, Washington, DC.

Weiss, A., Dorken, E., Nawab, S.H., "Model Variety Analysis: Determining the Need for Knowledge-Based Signal Processing," *Proceedings of the 1991 Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, CA, vol. 1, pp.185-89, Nov. 1991.

Appendix B

AI Journal Paper

IPUS: An Architecture for the Integrated Processing and Understanding of Signals

Victor R. Lesser* S. Hamid Nawab† Frank I. Klassner*

*Computer Science Department
University of Massachusetts
Amherst, Massachusetts 01003

†Electrical, Computer, and
Systems Engineering Dept.
Boston University
Boston, MA 02215

May 19, 1995

Abstract

The *Integrated Processing and Understanding of Signals* (IPUS) architecture is presented as a framework that exploits formal signal processing models to structure the bidirectional interaction between front-end signal processing and signal understanding processes. This architecture is appropriate for complex environments, which are characterized by variable signal to noise ratios, unpredictable source behaviors, and the simultaneous occurrence of objects whose signal signatures can distort each other. A key aspect of this architecture is that front-end signal processing is dynamically modifiable in response to scenario changes and to the need to *re-analyze* ambiguous or distorted data. The architecture tightly integrates the search for the appropriate front-end signal processing configuration with the search for plausible interpretations. In our opinion, this dual search, informed by formal signal processing theory, is a necessary component of perceptual systems that must interact with complex environments. To explain this architecture in detail, we discuss examples of its use in an implemented system for acoustic signal interpretation.

B.1 Introduction

Since the middle 1970's, a major focus in perceptual architecture design has been the identification and organization of knowledge to permit recovery from uncertainty introduced by front-end numeric signal processing algorithms (SPAs). One can categorize research efforts in this area along five dimensions according to where they emphasize the placement of this knowledge:

1. within high-level interpretation knowledge sources (HLKSs) (e.g., as improved or approximate models of environmental phenomena [18, 20, 32, 44]),
2. within numeric-level KSs (SPAs) (e.g., as control parameter optimization processes or feedback loops [8, 22, 42]),
3. in the control of HLKSs' application (e.g., in planning architectures for controlling KS activation and sophisticated evidential representations [6, 10, 11, 20, 39]),
4. in the control of SPAs' application (e.g., as differential diagnosis rules for SPA application to disambiguate objects in the environment [14, 15, 29] or as compiled "SPA trees" learned for particular objects [19]), and
5. in the control of the interaction between HLKSs and SPAs [1, 2, 5, 14, 15, 29, 23].

Over the past two decades, research efforts along each of the first four dimensions has been quite fruitful, yielding significant architectural paradigms. However, we believe that some of the assumptions made in these efforts have resulted in a paradigm not well suited to the perception of complex environments. Such environments are characterized by variable signal-to-noise ratios, unpredictable source behavior, and the simultaneous occurrence of objects whose signal signatures can mask or otherwise distort each other.

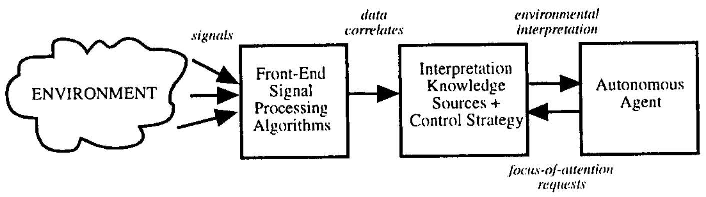

ENVIRONMENT
signals
Front-End Signal Processing Algorithms
data correlates
Interpretation Knowledge Sources + Control Strategy
environmental interpretation
Autonomous Agent
focus-of-attention requests

Figure B.1.1: *Classic Knowledge-Based Signal Processing Architecture. This paradigm imposes a unidirectional control flow that limits interpretation processes' analysis to only the single set of observations afforded by the fixed signal processing. Interpretation processes do not usually provide structured feedback to the front-end about either the adequacy of the signal processing outputs to be interpreted or any anticipated signal behavior.*

Consider the architectural paradigm in Figure B.1.1, which has usually been assumed by research efforts lying along the first four dimensions. It assumes that fixed signal processing in the front-end can provide adequate (not necessarily optimal) evidence for reliable interpretations regardless of the range of possible scenarios in the environment. In our opinion, this assumption is plausible for architectures that monitor stable environments, but not for those that monitor complex environments. In these environments, the choice of front-end SPAs is crucial to the generation of adequate evidence for interpretation processes. Parameter values inappropriate to the current scenario can render a perceptual system unable to interpret entire classes of environmental events correctly. Front-end SPA sets for complex environments must be dynamically modifiable to respond to scenario changes and to reprocess ambiguous or distorted data. "Dynamically modifiable" refers both to the ability to change SPA control parameter values and to the ability to select entirely new sets of front-end SPAs.

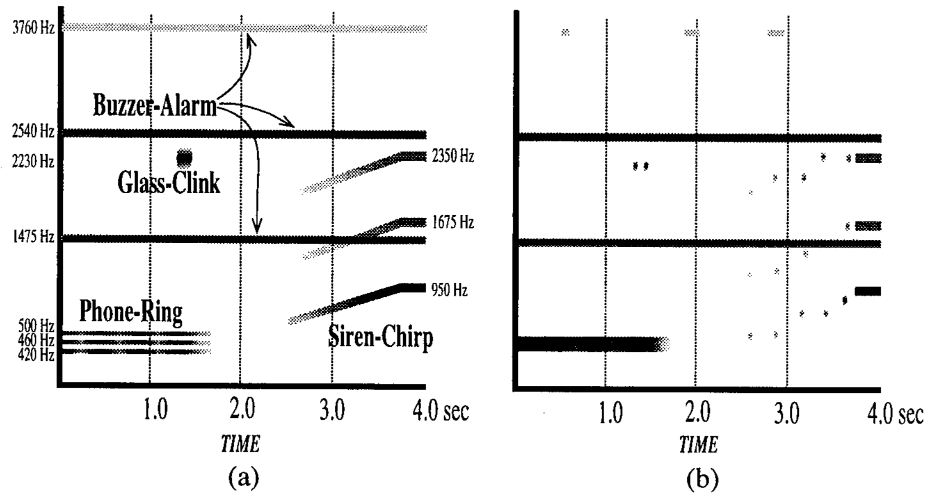


Figure B.1.2: *Figure B.1.2b shows distortions introduced by a STFT SPA and a peak-picker SPA with inappropriate parameter settings applied to the acoustic scenario described in Figure B.1.2a. Darker shading indicates higher energy. The STFT parameter settings used throughout B.1.2b were FFT-SIZE: 512, WINDOW-LENGTH: 512, and DECIMATION: 512, while the peak-picker's parameter setting was PEAK-THRESHOLD: 0.09. The signal was sampled at 8KHz. DECIMATION is the separation between consecutive analysis window positions; the value was set to 512 to permit the fastest possible processing of the data. PEAK-THRESHOLD is the normalized energy required for a discrete Fourier transform point to be considered as a peak. In B.1.2b's first second, Phone-Ring's tracks are merged because the STFT's frequency resolution is not adequate for such close features. Glass-Clink's frequency track is not even detected in B.1.2b's next second because the STFT's analysis window doesn't provide adequate time resolution to isolate the source's spectral features. The energy threshold causes the peak-picker to miss Buzzer-Alarm's low-energy track.*

Figure B.1.2 illustrates the utility of dynamically modifiable SPAs to interpret a complex acoustic environment. Figure B.1.2a shows the frequency tracks of four sound sources as they would appear if they were processed with Short-Time Fourier Transform [36] (STFT) SPAs appropriate for each portion of the scenario. Figure B.1.2b shows how the tracks appear when the entire scenario is processed by one STFT SPA appropriate only for the steady-state portion of the last sound in the scenario. Due to inappropriate processing, the first two seconds' analyses contain several distortions that would lead to ambiguous interpretations and completely undetected sources (see Figure B.1.2's caption).

These observations have led us to focus our work along the fifth knowledge-placement dimension: controlling HLKS/SPA interaction. Since the late 1980's, there have been several efforts to design architectures allowing interpretation processes to reconfigure signal processing. However, these architectures' processing/interpretation interactions have tended to be informal or domain-specific (see Section B.5).

In this paper we present the *Integrated Processing and Understanding of Signals* (IPUS) architecture as a formal and domain-independent framework for structuring HLKS/SPA interaction in complex environments [27, 28, 30, 31, 34, 35]. It enforces structured, bidirectional interaction between a perceputal system's interpretational components and signal processing components. This interaction combines the search for front-end SPA configurations appropriate to the environment with the search for plausible interpretations of front-end processing results. The architecture is instantiated by a domain's formal signal processing theory. It has four primary components as conceptual "hooks" for organizing and applying signal processing theory: discrepancy detection, discrepancy diagnosis, differential diagnosis, and signal reprocessing. These components have the following functionality:

- detect discrepancies between data expectations and actual data observations,
- diagnose these discrepancies and ascribe reasons for observational uncertainty,
- determine reprocessing strategies for uncertain data and expected scenario changes, and
- determine differential diagnosis strategies to disambiguate data with several alternative interpretations.

This paper discusses the generic IPUS architecture and its instantiation for acoustic signal interpretation. Acoustic signal interpretation in itself is an interesting problem that arises in applications such as assistive devices for the hearing impaired and robotic audition.[1] In the following sections we (1) discuss perception in complex environments (2) present motivations for the IPUS framework,

[1]The problem of identifying and tracking sounds.

(3) describe the generic IPUS architecture, (4) discuss related work, (5) describe an IPUS-based acoustic interpretation testbed, (6) illustrate the testbed's behavior using Figure B.1.2's scenario, (7) discuss the architecture's implications for SPA design, and (8) indicate directions for our future research.

B.2 Perception in Complex Environments

In this section we discuss relationships between the nature of perception in complex environments and the means by which systems actually perceive environments. In particular, we establish terminology for describing environments and for discussing context-dependent suitability of SPAs. We represent environments using the following definitions.

Definition 1 (Environment) *An environment is a triple ($\mathcal{O}$,$\mathcal{F}$,$\mathcal{R}$) where $\mathcal{O}$ is the set of observable objects, $\mathcal{F}$ is the set of all features that can be used to describe objects, and $\mathcal{R}$ is a set of context rules describing how features interact with each other when more than one object is being perceived in the environment.*

Definition 2 (Objects) *Each object belongs to a unique object class. Object classes are defined by sets of feature descriptions. Each set specifies a subset of features from $\mathcal{F}$ and ranges of permissible values for these features. An object is an instance of an object class if its feature values lie within a descriptor set of the class.*

Definition 3 (Contexts) *A context is the set of all specific objects, with their orientation, observed in an environment. A permissible context is defined as a set of objects which are permitted to co-occur. Unless otherwise proscribed by the specific application domain, a permissible context may contain several instances of the same object class.*

In audition, the orientation of an object includes domain-dependent characterizations such as distance, loudness, and velocity. In another domain such as vision, orientation would include characterizations such as pose, distance, and velocity.

Definition 4 (Context Rules) *A context rule is a pair (C,F). C is a permissible context and $F \subseteq f_{obj} \times f_{env}$. Here f_{obj} is the union of instantiated features from all the objects in C, and f_{env} is a powerset of $\mathcal{F}$ with instantiated values. The set F indicates the observability of the objects' instantiated features when they are considered in the context C. Elements in F of the form $\{f_1, \{f_1\}\}$ indicate the instantiated feature f_1 is observable in the context; elements of the form $\{f_1, \{g_1, \ldots, g_n\}\}$ indicate the instantiated feature is masked or otherwise distorted to appear as different instantiated feature(s) $\{g_1, \ldots, g_n\}$ from f_{env}. Note that f_x indicates a feature and its particular value.*

The rules indicate how the features of co-occurring objects interact with each other without regard to how their signals are processed. For example, such rules from vision would address the occlusion of objects by other objects, while such rules from audition would address the summed-energy of overlapping frequency components from multiple sounds. Definition 4 describes only the *kind* (not the *form*) of knowledge that perceptual systems should have about contexts. The definition's knowledge representation is combinatorially explosive and certainly could not be used in any real system.

Having defined our concept of a perceptual system's environment, let us now consider SPAs, the means by which a system processes the signals from its environment. There are two levels of abstraction for describing SPAs: generic SPAs and SPA instances. SPA instances are specified by specific values for a generic SPA's control parameters. Where there is no ambiguity in the discussion between generic SPAs and SPA instances, we will use the term "SPA" to refer to an SPA instance. When applied to signals, SPAs produce *correlates.* These are used as evidence to support hypotheses that particular features (not necessarily associated with any object) are present in the environment. We refer to the correlate set produced by an SPA as that SPA's *computed correlate set.*

An SPA's parameter values induce capabilities or limitations with respect to the scenario being monitored. Consider the generic Short-Time Fourier Transform (STFT) algorithm [36] in the acoustic domain. An STFT instance has particular values for its parameters, such as analysis window length, frequency-sampling rate, and decimation interval (separation between consecutive analysis window positions). Depending on assumptions about a scenario's spectral features and their time-variant nature, these parameter values increase or decrease the instance's usefulness in monitoring the scenario. An instance with a large window length will provide fine frequency resolution for scenarios containing sounds ("acoustic objects") with time-invariant components, but at the cost of poor time resolution for sounds with time-varying components.[2]

In complex environments, there are often many SPAs which can potentially compute a correlate's value. The effectiveness of an SPA to produce correlates that can support hypothesized object features is dependent in general upon the context in which the correlates are to be computed, the specific values of the object features, and the SPA's parameter values. We will consider an SPA's parameter values appropriate to a context if the SPA's correlates can provide not just support, but *unambiguous* support for all the features of all the objects in the context.

Figure B.2.3 uses sound disambiguation to show the relationship between context-dependent correlate computation and interpretation ambiguity more concretely. When analyzed in isolation, the hairdryer's two frequency tracks are unambiguously supported by the correlates from STFT-1. However, when the hairdryer's tracks are analyzed in conjunction with the telephone in the second context, ambiguity arises. The new tracks in Figure B.2.3b indicate the potential presence of a new

[2]A variant analysis of the Heisenberg Uncertainty Principle implies that one cannot obtain a STFT SPA instance (or, for that matter, design a new generic SPA) that simultaneously provides infinite frequency resolution and infinite time resolution.

sound that matches the telephone model except for its lowest frequency track. The hairdryer's lower-frequency track cannot be unambiguously supported by the same SPA's correlates, since at least some of the track's potential support could alternatively support the phone's low-frequency components. Fourier theory can attribute the ambiguity to the SPA's poor frequency resolution capabilities and indicate that the second context should be reanalyzed by a more appropriate SPA. When the second context's signal is analyzed by STFT-2, the SPA's finer resolution confirms this explanation for the ambiguity and provides correlates that unambiguously support both the hairdryer's *and* the telephone's tracks.

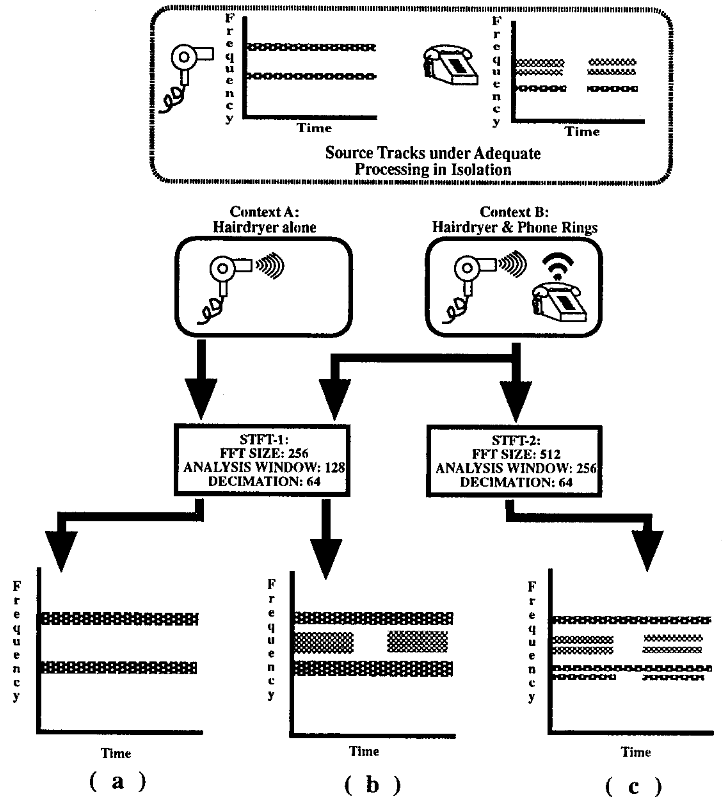

Frequency
Time
Frequency
Time
Source Tracks under Adequate Processing in Isolation
Context A: Hairdryer alone
Context B: Hairdryer & Phone Rings
STFT-1: FFT SIZE: 256 ANALYSIS WINDOW: 128 DECIMATION: 64
STFT-2: FFT SIZE: 512 ANALYSIS WINDOW: 256 DECIMATION: 64
Frequency
Time
(a)
Frequency
Time
(b)
Frequency
Time
(c)

Figure B.2.3: *Context-Dependent Correlate Computation. When STFT-1 analyzes context A's signal, its frequency correlates in (a) are adequate for unambiguously identifying the hairdryer's two frequency tracks. When the same SPA analyzes context B's signal, however, its frequency correlates in (b) are not adequate for unambiguously supporting the hairdryer's two tracks AND the phone's three tracks. Context B's signal requires processing by STFT-2 with a finer frequency resolution in order to produce correlates in (c) that unambiguously support the two sources' tracks.*

At this point we see that to select SPA instances appropriate to a particular scenario, a perceptual system must consider the features corresponding to the input signal. This leads to the apparent circularity that choosing appropriate SPA parameter values requires knowledge about the signal, but this knowledge can only be obtained by first processing the signal with an SPA with appropriate parameter settings. Thus, in complex environments the search for appropriate interpretations must be intimately connected with the search for appropriate SPA instances.

The features that perceptual systems can monitor in complex environments fall into two classes. The first class contains features which can be used to indicate the existence of one or more objects, though not necessarily the objects' identities. These features often have supporting correlates that can be computed independent of the context being analyzed. In the auditory domain, for example, any collection of one or more "sound objects" may be conceptualized as an acoustic intensity distribution with minimum and maximum limits on gross features such as temporal spread, frequency spread, duration of silence intervals, and degree of randomness in intensity fluctuations. Such gross features' correlates can generally be computed in a context-independent manner; hence we call them *context-independent features.*

The second feature class contains those features which can be used to identify an object or track the behavioral changes of an object. The computation of correlates to support these features is often very sensitive to the context being analyzed; hence we call them *context-dependent features.* In the auditory domain, for example, a frequency track would be a context-dependent feature of a sound ("acoustic object"). If the current scenario has no sounds besides the sound containing a particular track T_0, then an STFT with parameters providing only very coarse frequency resolution would still produce correlates that could support the track's existence. Now assume that the current scenario changes so that there are other sounds in the environment with frequency tracks $T_1, \ldots, T_n$. In this new scenario only STFTs providing frequency resolution of at least the minimum difference between T_0's frequency and the other tracks' frequencies would produce correlates that could unambiguously support T_0's existence.

It is important to note that the distinction between context-independent and context-dependent features lies in the features' usage. If a feature is used only to indicate the *presence* of some object(s), the feature is considered context-independent. However, if the same feature were to be used as support for the *identity* of some object(s), it would in general require context-dependent correlate computation, and would therefore be considered a context-dependent feature.

This section's discussion about complex environments and the basic means for analyzing their signals serves as background for Section B.3. The focus in that section is on how a domain's signal processing theory can be used to guide the design of an architecture for controlling the process of SPA application.

B.3 Architectural Motivation

Past research efforts within the traditional paradigm for perceptual system design (Figure B.1.1) have produced architectures that require the identification of a set of features and SPAs applicable to all scenarios the environment may produce. This requirement is feasible only for significantly constrained environments. Under the traditional paradigm, complex environments can require combinatorially explosive SPA sets with multiple parameter settings to capture the variety of signals adequately [17] and to handle the variety of processing goals the current scenario may dictate. As an example of variable processing goals, consider a system with the primary goal of responding to either the sounds of an infant or a ringing telephone while ignoring other sounds. This may be done by monitoring a medium-frequency band. If an infant sound is detected, the system's goal may then switch to determining whether the infant is crying or choking while ignoring telephone rings. Such a goal might then be accomplished by switching to lower-frequency spectral regions with specialized SPAs.

To circumvent the combinatorial explosion, one could reason that a small SPA set might be sufficient if comparisons could be made between the SPAs' computed correlates and dynamically-generated formal expectations. We use the term *anticipated correlate set* to refer to the set of expectations about an SPA's computed correlate set. Any computed correlates whose coordinates and values do not match those of any anticipated correlates are considered unanticipated. Unmet SPA output expectations can indicate that either the expectations are based on incorrect interpretations or that the SPA's computed correlates have been distorted because the SPA's parameter values are inappropriate to the current scenario. In the first case a perceptual system could re-interpret the current scenario based on the SPA's correlates, while in the second case a perceptual system could reconfigure the SPA's parameters or replace it with a more appropriate SPA. The important assumption in this solution is that there is a basis for generating the expectations, detecting the unmet expectations, and deciding between the two possible classes of explanations for the unmet expectations. We argue that a domain's *formal signal processing theory* can play this role.

An SPA's correlates can be compared with expectations based on object models or on *a priori* environment constraints such as maximum bounds on sounds' rate of temporal change in frequency. Referring back to our assumption about rules for the interaction of co-occuring objects' features, these "context rules" could also provide a basis for checking SPA appropriateness. Most importantly, a domain's signal processing theory can specify how one SPA's correlates for a context-independent feature can serve as the basis of expectations for another SPA's output correlates. This specification can serve to check an SPA's appropriateness to the environment. It can also serve to decide where to selectively apply another SPA in the signal data stream to obtain correlates for context-dependent features.

Figure B.3.4 illustrates these concepts with an example from the acoustic processing of footsteps in a noisy environment. The example uses two complementary generic SPAs: a time-domain energy tracker and an STFT. The time-domain energy tracker detects a short, uniform energy burst that should correspond to short tracks in the frequency domain, according to acoustic signal processing theory. When analyzed by STFT-1 with its wide analysis window, the footstep's impulsive energy is smoothed with surrounding noise and fails to appear as a short frequency track in the STFT's correlates. In other words, the STFT's correlates are subject to a smoothing distortion. The temporal locations and durations of the energy tracker's energy bursts serve two purposes. First, they indicate that STFT-1 was potentially inappropriate to the current environment. Second, they serve as the basis for generating STFT-2 with a narrower analysis window and smaller time decimation interval to apply to the region in the signal where a new source is suspected. This STFT's correlates not only confirm the belief that the first STFT was inappropriate to the environment but also more strongly support the existence of the impulsive footsteps than the energy tracker's correlates did by themselves.

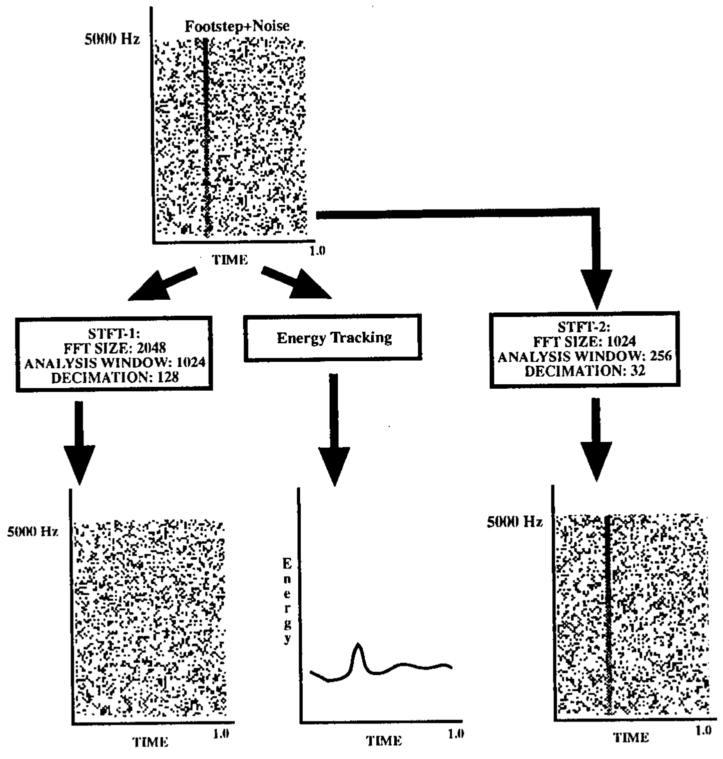


Figure B.3.4: *Context-Dependent Correlate Computation. The energy tracking SPA provides correlates for context-independent energy burst features. These features which guide the focused application of an STFT with parameters to find frequency-track correlates for the footstep impulse in a noisy environment.*

The preceding example provides instances of three generic roles that a domain's formal signal processing theory can play in guiding interpretation and processing in a complex environment:

- provide methods to determine discrepancies between an SPA's expected correlate set and its computed correlate set.

- define distortion processes that explain how discrepancies between expectations and an SPA's computed correlates result when the SPA has inappropriate values for specific parameters.
- specify new strategies to reprocess signals so that distortions are removed or ambiguous data is disambiguated.

These observations about the power of formal signal processing theory in analyzing complex environments lead to our decision to incorporate a "discrepancy detection, diagnosis, and reprocessing loop" as the backbone of the IPUS architecture. We believe that the explicit representation of the knowledge in signal processing theory is crucial to systems that monitor complex environments. Our design of IPUS is motivated by the thesis that complex environments require dynamic, context-dependent feature selection concurrent with dynamic, context-dependent selection of appropriate SPAs for extracting correlates to support the features. The goal of the framework is to use theoretical relationships between SPA parameters and SPA outputs to structure the dual searches for SPAs appropriate to a scenario and for interpretations appropriate to the SPAs' correlates.

B.4 Generic IPUS Architecture

This section has three parts. The first part presents a summary of the architecture. The second part discusses the generic specifications of each component of the architecture's reprocessing loop: discrepancy detection, discrepancy diagnosis, reprocessing, and differential diagnosis. The third part describes the architecture's control framework. Section B.6.1 provides summaries of the algorithms used to instantiate the IPUS components in the acoustic interpretation testbed [31].

B.4.1 Architecture Summary

The generic IPUS architecture, with its primary data and control flow, appears in Figure B.4.5a. Figure B.4.5b shows its instantiation in the acoustic interpretation testbed to be discussed in Section B.6.2. Two types of signal interpretation hypotheses are stored on the hierarchical blackboard: interpretations of correlates from current and past signal analyses, and expectations about the interpretations of data correlates from future analyses.

Our design of the IPUS framework assumes that signal data is submitted for analysis a block at a time. IPUS uses an iterative process for converging to the appropriate SPAs and interpretations. For each block of data, the loop starts by processing the signal with an initial configuration of SPAs. These SPAs are selected not only to identify and track the signals most likely to occur in the environment, but also to provide indications of when less likely or unknown signals have occurred.

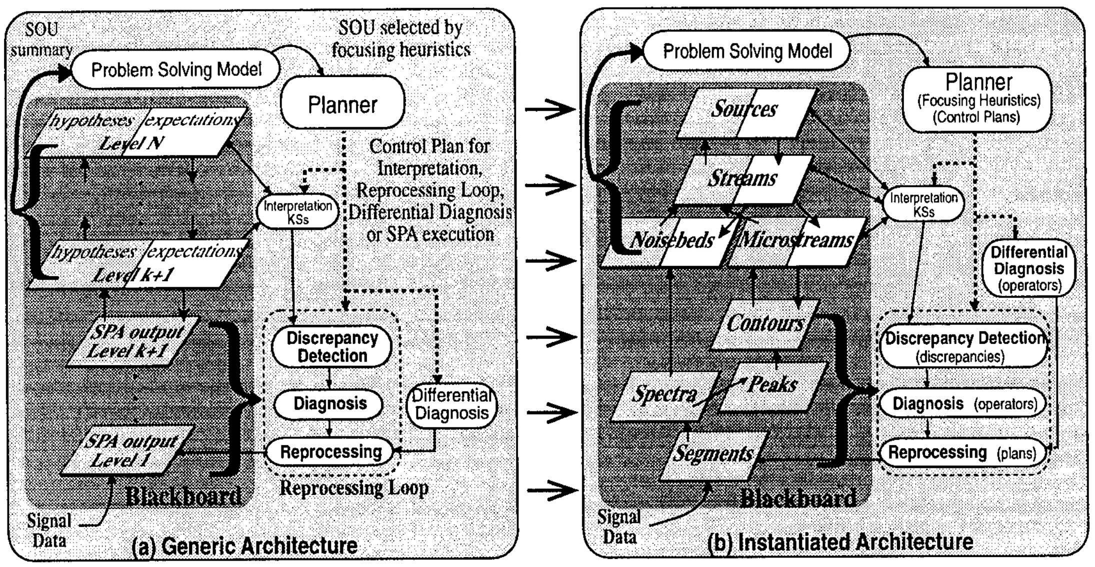


Figure B.4.5: *Figure B.4.5a shows the generic IPUS architecture and Figure B.4.5b shows the architecture instantiated for the sound understanding testbed. Solid arrow lines indicate dataflow relations. Dotted arrow lines indicate classes of plans that the planner can pursue when trying to reduce or eliminate particular uncertainties (discrepancies) in the problem solving model that were selected by the focusing heuristics. Parenthesized terms indicate knowledge added to the planner or system knowledge sources to instantiate the architecture for an application. Note that reprocessing plans can cause SPA execution at any SPA output level, not just the lowest.*

In the next part of the loop, a *discrepancy detection* process tests for discrepancies between the correlates of each SPA in the current configuration and (1) the correlates of other SPAs in the configuration, (2) application-domain constraints, and (3) the correlates' anticipated form based on high-level expectations. Architectural control permits this process to execute both after SPA output is generated and after interpretation problem solving hypotheses are generated. If discrepancies are detected, a *diagnosis* process attempts to explain them by mapping them to a sequence of qualitative distortion hypotheses. The loop ends with a *signal reprocessing* stage that proposes and executes a search plan to find a new front-end (i.e., a set of instantiated SPAs) to eliminate or reduce the hypothesized distortions. After the loop's completion, if there are any similarly-rated competing top-level interpretations, a *differential diagnosis* process selects and executes a reprocessing plan to find correlates for features that will discriminate among the alternatives.

Although the architecture requires the initial processing of data one block at a time, the loop's diagnosis, reprocessing, and differential diagnosis components are not restricted to examining only the current block's processing results. If the current block's processing results imply the possibility that earlier blocks were misinterpreted or inappropriately reprocessed, those components can be applied to the earlier blocks as well as the current blocks. Additionally, reprocessing strategies and discrepancy detection application-constraints tests can include the postponement of reprocessing or discrepancy declarations until specified conditions are met in the next data block(s).

B.4.2 IPUS Reprocessing Loop Components

This section discusses the generic specifications of each component of the architecture's reprocessing loop, as depicted in Figure B.4.5a.

Discrepancy Detection

The discrepancy detection process is crucial to the IPUS architecture's iterative approach. Our specification of the process requires it to recognize three groups of discrepancies, based on the source of the anticipated correlates used in the comparisons.

fault A discrepancy between an SPA's computed correlates and correlates from other SPAs applied to the same signal data. This class is included based on two propositions. The first is that correlates for context-dependent features, if computed by SPAs appropriate to the context, do not contradict the correlates for context-independent features. The second is that correlates for context-dependent features, if computed by SPAs appropriate to the context, do not contradict other context-dependent correlates computed by other SPAs from the same data. As an example, refer to Figure B.3.4 where the energy tracking SPA indicates a short burst of energy while the first STFT's correlates do not support new frequency tracks during the burst's time period. A fault should be declared since Fourier theory requires the burst's presence in both analyses, given the assumption that the STFT analysis was appropriate to the context.

violation A discrepancy detected between an SPA's computed correlates and domain constraints. This class is included based on the proposition that correlates, if computed by SPAs appropriate to the context, do not support features that violate the environment's physical constraints. As an example, if the application domain is considered subject only to wideband gaussian noise (5000 Hz wide), STFT output correlates showing only a narrowband noise signal (say 500 Hz wide) would give rise to a violation. Note that violations can indicate either that an SPA was inappropriately applied or that the environment's characteristics have changed from those in the original definition. In the first case reprocessing based on the environment's definition should succeed in eliminating the discrepancy. In the second case reprocessing based on the environment's (invalid) definition will fail. Failures of the second type are recorded as distortions to be expected due to environmental changes and prevent needless execution of the reprocessing loop when they are detected again.

conflict A discrepancy between an SPA's computed correlates and model-based expectations. Model-based expectations arise from two sources. The first source is the set of models for objects already assumed to be present. The second source is the set of models for objects under consideration for interpreting newly-detected correlates in the current block of data. Conflict discrepancies may involve either a total or a partial mismatch between correlates and the hypotheses they were supposed to support. This class is included based on the proposition that features supported by correlates computed from appropriate SPAs ought to be completely consistent with the object features specified by the context expected to be observed. "Object features" includes not only features that are not expected to be distorted but also features that are expected to be distorted because of the existence of other objects in the environment. Conflicts can indicate that an SPA is not appropriate to the context or that the context actually contained objects different from those expected. As a simple example, a conflict would occur when the interpretations of past correlates predict a sound with two sinusoids at 230 Hz and at 250 Hz with no decline in their amplitudes and current STFT correlates support one or none of the sinusoids. It could indicate that possibly the STFT's energy threshold is inappropriate because the sound's volume decreased, or that a new sound is masking the expected sound. Because we make expectations take on the maximum possible values for their object features, this conflict could also indicate that the expectation's duration was too long.

Examination of a wide range of domains reveals two generic classes of correlates: *point correlates* and *region correlates.* A point correlate is a value associated with one point in the SPA output coordinate space. A region correlate is a value associated with a subset of the SPA output space. Consider the following examples. A spectral peak energy value in the "time, frequency, energy" space of acoustic signal processing and an image pixel intensity value in the "x, y, intensity" space of image processing are examples of point correlates. A noise-distribution tag for a region in a radar sweep and a mean-intensity value for a region in the output of an image filtering SPA are examples of region correlates. A track of spectral peaks over time from a series of FFT analyses is an example of a region correlate comprised of non-contiguous subsets of the SPAs' output space.

For both point and region correlates, we require that the IPUS discrepancy detection component be able to check for the following generic discrepancies between an SPA's anticipated correlate set and its computed correlate set.

1. **missing:** An anticipated correlate is not in the computed correlate set. An example of this discrepancy in the acoustic domain occurs when a spectral peak is expected in the output of an FFT SPA, but is not found.

2. **unassociated:** An unanticipated correlate occurs in the computed correlate set. An example of this discrepancy in the radar domain occurs when an unanticipated clutter region is produced during a radar sweep.

3. **value-shift:** A correlate is found in the computed correlate set at its anticipated coordinates, but with an unanticipated value. In the visual domain we encounter this discrepancy when an image region's hue label produced by an intensity analysis SPA is brighter than expected.

4. **coordinate-shift:** A correlate with an anticipated value is found in the computed correlate set but at unanticipated coordinates. This includes the situation where a region's boundaries shift from their expected locations. An example of this discrepancy in the acoustic domain occurs when a track of spectral peaks produced by a curve-fitting algorithm has the correct energy value but is 30 Hz from its expected position.

5. **merge:** Two or more anticipated correlates are deemed to have appeared as one unanticipated correlate in the computed correlate set. The criteria for this merging are domain-specific and often depend on relationships between the missing correlates' values or coordinates and the unanticipated correlate's value or coordinates. An example of this discrepancy in the visual domain occurs when two adjacent regions with different expected textures are replaced by one region with an unanticipated texture.

6. **fragmentation:** An anticipated correlate is deemed to have been replaced by several unanticipated correlates in the computed correlate set. The criteria for this splitting are domain-specific and often depend on relationships between the missing correlate's values or coordinates and the unanticipated correlates' values or coordinates. An example of this discrepancy in the radar domain occurs when a noise-analysis SPA computes two or more small regions with a particular noise-distribution label instead of an expected single region with that label.

Discrepancy Diagnosis

A domain's formal signal processing theory can predict the form computed correlates will take not only when an SPA is applied with parameter values appropriate to the context, but also when an SPA is applied with inappropriate parameter values. We relate a signal processing theory's content to SPAs and their interaction with the environment in terms of *SPA processing models.* An SPA processing model describes how the output of the SPA changes when one of its control parameters is varied while all the others are held fixed.

SPA processing models serve as the basis for defining how the parameter settings of an SPA can introduce distortions into the SPA's computed correlates. These distortions cause correlate discrepancies. Consider an SPA processing model corresponding to the STFT's WINDOW-LENGTH parameter and how this model can be used to define distortions. Refering to Figure B.2.3, as this parameter's value increases, merged and missing correlate discrepancies disappear. Conversely, as the parameter's value decreases, merged and missing correlate discrepancies occur more frequently. Formally, assume that an STFT with an analysis window of W sample points is applied to a signal sampled at R samples per second. If the signal came from a scenario containing frequency tracks closer than R/W Hz, Fourier theory predicts that the tracks will be merged in the STFT's computed correlates.

When discrepancies are detected, *diagnosis* can be performed to obtain an "inverse" mapping from the discrepancies and to qualitative hypotheses that explain them in terms of distortions. This diagnosis process relies on an environment's context rules and the domain's SPA processing models to define distortion processes that take place when an SPA's assumptions about its input signals are violated [37]. Note that there is a difference between discrepancies and signal distortion processes. Distortion processes are used to explain discrepancies. It is also possible for several distortion processes to explain the same kinds of discrepancies. A "low frequency resolution" process explains the 'missing' and 'unassociated' discrepancies in Figure B.2.3's example, and a "low time resolution" process explains the 'missing' discrepancy in Figure B.3.4's example.

As another simple diagnostic example, consider the conflict discrepancy where frequency components previously observed at 225 Hz and 250 Hz "disappear" from the current STFT output but a "new" component is observed midway between the original components' positions. The STFT processing models provide us with the concept of a "low frequency resolution" distortion process which can account for the missing and unanticipated correlates in the STFT output. In discrepancy diagnosis, this specific distortion's definition would serve as the basis for checking if it is plausible that the two components may have drifted too close to each other for the current STFT instance to be able to resolve them. If this is indeed plausible, the distortion process explains the presence of just a single component in the current STFT output.

Reprocessing and Differential Diagnosis

The *signal reprocessing* component uses explanations from the diagnosis component to propose and execute search plans for finding new SPA control parameter values that eliminate or reduce the hypothesized distortions. In the course of a reprocessing plan's execution, the signal data may be reprocessed several times under different SPAs with different parameter values. The incremental search is necessary because the diagnosis explanation is at least partially qualitative, and therefore it is generally impossible to predict *a priori* exact parameter values to be used in the reprocessing. The reprocessing component relies on SPA processing models to select new SPAs and/or parameter values when instantiating the proposed reprocessing plan. Continuing the frequency resolution example from the previous subsection, the STFT processing model's quantitative relationship between parameter values and correlate output would indicate the need for a STFT instance with a longer analysis window for obtaining better frequency resolution.

In the course of processing signal data, IPUS-based systems will encounter signals that could support several alternative interpretations. In addition to natural similarities among several objects' features, ambiguous sets of alternative interpretations can also arise from co-occuring objects' interactions and from applying SPAs inappropriate to a context. The differential diagnosis component implements what we have previously referred to in Section B.3 as the dynamic, context-dependent selection of features to disambiguate objects. It uses SPA processing models to predict how the front-end SPAs' parameter values could have made correlates for different features of alternative objects appear similar. Based on these predictions, the reprocessing component can then propose a reprocessing strategy to disambiguate the features' correlates.

The dual search in IPUS becomes obvious with the following two observations. Each time the data is reprocessed, whether for disambiguation or distortion elimination, a new state in the SPA instance search space is examined and tested for how well it eliminates or reduces distortions. At the same time, the distortion elimination or disambiguation measurement is predicated on the assumption that the system's current state in the interpretation space matches the actual context being observed. We will see later in Section B.7.2 that failure to remove a hypothesized distortion after a bounded search in the SPA instance space will often lead to a new search in the interpretation space. This happens based on the following reasoning. The diagnosis and reprocessing results represent an attempt to justify the assumption that the current interpretation is correct. If either diagnosis or reprocessing fails, there is a strong likelihood that the current interpretation is not correct and a new search is required in the interpretation space. Furthermore, the results of failed reprocessing can constrain the new interpretation search by eliminating from consideration objects with features requiring correlates that should have been found during the reprocessing.

B.4.3 Control in IPUS

Depending upon the class(es) of discrepancies detected and the context in which interpretation is being carried out, an IPUS-based system can use different strategies to resolve (i.e. explain and possibly eliminate) the discrepancies. For example, in a situation where real-time processing deadlines are tight, the system may not even attempt to resolve conflict discrepancies involving minor mismatches in order to conserve time. In a situation where time is costly but not prohibitive, however, the system may decide to engage the diagnostic process on the discrepancy, but then to forego actual reprocessing of the signal because the proffered explanation would require reprocessing a set of data too large to be accommodated by the time constraints. That is, for this case the system may decide that the successful *generation* of an explanation alone is sufficient to resolve the discrepancy. Finally, in a non-time-critical situation or when analyzing data from an important source, the system may decide to engage the diagnostic process and reprocess the data on the basis of the explanation in order to verify the explanation's plausibility as part of resolving the discrepancy.

We designed IPUS to serve as the basis of systems for producing perceptual interpretations with acceptable uncertainty levels. Therefore, we had to provide the architecture's control framework with a formalism for representing factors that affect interpretations' confidence levels. The control framework also had to support context-sensitive focusing on particular uncertainties in order to control engagement and interruption of the architecture's reprocessing loop.

For these reasons, IPUS uses the RESUN [11, 12] framework to control knowledge source (KS) execution. This framework supports the view of interpretation as a process of gathering evidence to resolve hypotheses' sources of uncertainty (SOUs). It incorporates a language for representing SOUs as structures which trigger the selection of appropriate interpretation strategies. Problem-solving is driven by information in the *problem solving model*, which is a summary of the current interpretations and the SOUs associated with each one's supporting hypotheses. An incremental, reactive planner maintains control using *control plans* and *focusing heuristics.* Control plans are schemas that define the strategies and SPAs available to the system for processing and interpreting data, and for resolving interpretation uncertainties. Focusing heuristics are context-sensitive tests to select SOUs to resolve and processing strategies to pursue.

The RESUN framework endows IPUS with two basic problem-solving modes: *evidence aggregation* and *differential diagnosis.* Evidence aggregation problem solving seeks data for increasing or decreasing the certainty of one particular interpretation, whereas differential diagnosis problem solving seeks data for resolving ambiguities that produced competing interpretations. Through these problem solving approaches, IPUS-based systems can decide when to reprocess data previously examined under one SPA with another SPA to obtain evidence for resolving uncertainties.

The RESUN framework was developed to address current interpretation systems' limited ability to express and react to the reasons for interpretation hypotheses' uncertainty. It emphasizes the separation of hypothesis belief evaluation from control decision evaluation by making control responsive not only to the levels of numeric belief in hypotheses but also to the presence of specific SOUs in the problem-solving model. The control plan formalism supports opportunistic control through a refocusing mechanism that lets the planner switch among several plan elaboration points (current leaf nodes in the plan tree) in a context-dependent manner. It also permits reprocessing strategies to be expressed as alternative control plans, which are selected on the basis of SOUs describing discrepancies and their explanations.

B.5 Related Work

The IPUS architecture explores how formal signal processing knowledge such as Fourier theory can be organized and applied in the fifth of the knowledge-placement dimensions discussed in Section B.1. This research represents the formalization and extension of concepts explored in earlier work on a diagnosis system that exploited formal signal processing theory to debug signal processing systems [37] and in work on meta-level control [24, 25] that used a process of fault-detection, diagnosis, and replanning to decide the most appropriate parameters for controlling a problem-solving system.

Although we oriented this research most strongly along the fifth knowledge-placement dimension, we feel it has implications for work along the other four dimensions as well. The architecture supports the use of an application domain's formal signal processing theory in selecting approximate or specialized SPAs for context-dependent application to specific portions of a signal [33]. For this reason the research also extends work that emphasizes the fourth dimension (control of SPA application).

Several recent systems have been developed that provide for structured interaction between interpretation activity and numeric-level signal processing. In this section we discuss selected frameworks or systems as representatives of general approaches to the problem of controlling the interaction of signal processing and environmental interpretation in perceptual systems. The general approaches are described in terms of the IPUS components they functionally include.

The perceptual framework of Hayes-Roth's GUARDIAN system [23] is typical of systems whose input data points already represent useful information and require little formal front-end processing other than to control the rate of information flow. The system incorporates an input-data management component that controls the sampling rate of signals in response to workload constraints. Information flow is controlled through variable sample-value thresholds and variable sampling rates. This control framework is somewhat limited since it is based only on the system's time requirements for reasoning about classes of signals, and provides good performance primarily because the signals monitored are relatively simple and noise-free in nature: heart-rate, temperature fluctuations, etc. The framework's lack of centralized components for any of the four IPUS tasks leads to inadequate generality for the wide range of signals-environment interactions which can include signals containing complex structures that must be modeled over time in the presence of variable noise levels. Note that we are not implying that frameworks in this class do not perform any diagnostic reasoning. We are only observing that this reasoning capability is not applied to the identification of potentially adverse interactions between the environmental signal and the front-end processing.

Dawant's framework [14] is closer in spirit to IPUS. It is typical of systems designed with the intent of providing alternative evidence sources as "backup" evidence when moderate deviations are observed between signal behavior and partially-matched signal event models. The framework does not support the selective reprocessing or selective application of specialized SPAs since data is always gathered from every front-end SPA whether required for interpretation improvement or not. This reliance on a fixed set of SPAs (regardless of whether their control parameters are variable) that are all always executed leads to systems where more and more SPAs are added to front-ends as the environmental complexity increases, ending in a combinatorial explosion in the number of SPAs necessary to unambiguously identify all signals in an environment. Unlike IPUS, most architectures in this category operate on the implicit assumption that the signal-generating environment will not interact adversely with the signal processing algorithms' limitations to produce output distortions that might not have occurred if more appropriate processing algorithms had been used. Any deviations between observed signal behavior and available signal event models are attributed to chance variations in the *source* being monitored, never to the signal's *interaction* with inappropriate SPAs or with other sources in the environment.

De Mori et al. [15] developed a formal interaction framework in a system to recognize spoken letters of the English alphabet. This framework is representative of architectures with strong reliance on differential diagnosis techniques. These architectures are often employed in domains where there is little or no dependence between consecutive signal events. Interpretations in the system were generated by learned rules expressing letter identifications in terms of a signal-event grammar. Often more than one letter could be indicated by a single rule (in their terminology the rule has a *confusion set*). When such rules are activated, the system pursues a differential diagnosis strategy relying on rules describing SPAs that are suited to disambiguating confusion sets with given members. Thus, the system makes use of selective SPA application and differential diagnosis strategies. However, given the framework's relatively restricted application domain, there is a serious question of whether the approach can be scaled up without including the ability to model the environment's signal processing theory. Since the environment of the system considers its objects (letters) as isolated, unrelated entities, the framework does not incorporate any use of diagnosis in conjunction with environmental constraints (e.g., A 'C' has been identified at time t_{-1} and a 'B' is expected at time t_0 since there is an environmental constraint that 'B's follow 'C's. No behavior supporting the expectation is observed, so diagnostic reasoning should be attempted to explain why).

GOLDIE [29] is an image segmentation system that uses high-level interpretation goals to guide the choice of numeric-level segmentation algorithms, their sensitivity settings, and region of application within an image. The system's architecture represents the set of architectures that place strong emphasis on selective SPA application without explicit guidance from formal signal-processing theory. The system uses a "hypothesize-and-test" strategy to search for algorithms that will satisfy high-level goals, given the current image data. While it incorporates an explicit representation of algorithm capabilities to aid in this search, and an explicit representation of reasons for why it assumes an algorithm is appropriate or inappropriate to a particular region, the system notably does not incorporate any diagnosis component for analyzing unexpected "low quality" segmentations. If an algorithm were applied to a region and the resulting segmentation were of unexpectedly low quality, the framework would not parallel IPUS and attempt to diagnose the discrepancy and exploit this information to reformulate the algorithm search but would select the next highest rated algorithm from the original search.

In the same category as GOLDIE is TraX [5], a system for interpreting image frame sequences. Although its design was driven by the goal of supporting multiple, concurrent object descriptions, the system incorporates some concepts similar to those in our formulation of the IPUS architecture. The system supports detection of deviations from expected measurements and determination of the possibility that these deviations might have resulted from processing techniques inappropriate to the current context. In a manner similar to conflict discrepancy detection in IPUS, TraX compares higher-level expectations from previous frames against its segmentation SPAs' outputs for the current frame. In contrast to the IPUS architecture specification, however, TraX does not use models derived from an underlying theory for its SPAs to inform the discrepancy detection and diagnosis processes. It relies instead on empirically derived statistical performance models for the segmentation algorithms. While TraX allows for the use of different SPAs for different contexts, it does not support the adaptation of SPAs' control parameters for different contexts.

Bell and Pau [1, 2] formalize the search for processing parameter values in numeric-level image understanding algorithms in terms of the Prolog language's unification and backtracking mechanisms. They express SPAs as predicates defined on tuples of the form $(M, p_1, \ldots, p_n)$, where M represents an image pattern and the p's represent SPA control parameters. These predicates are true for all tuples where M can be found in the SPA output when its control values are set to the tuple's p values. Prolog's unification mechanism enables these predicates to be used in both goal-directed and data-driven modes. In a goal-driven mode, M is specified and some of the parameters are left unbound. The unification mechanism verifies the predicate by iteratively binding the unspecified parameters to values from a permissible value set, applying the SPA, then checking if the pattern is found. In a data-driven mode, M is not bound and the parameter values are set to those of the front-end processing. M is then bound to the SPA results. The method relies on Prolog's backtracking *cuts* [21] to limit parameter-value search. A cut is a point in the verification search space beyond which Prolog cannot backtrack. This reliance on a language primitive makes it difficult to explicitly represent (and therefore to reason about) heuristic expert knowledge for constraining parameter-value search as can be done in IPUS's reprocessing component. The cut mechanism also does not permit the use of formal diagnostic reasoning to further constrain parameter-value search based on the cause of an SPA predicate failure.

Research in active vision and robotics has recognized the importance of tracking-oriented front-end SPA reconfiguration [43], and tends to use a control-theoretic approach for making reconfiguration decisions. It is indeed sometimes possible to reduce the reconfiguration of small sets of front-end SPAs to problems in linear control theory. In general, however, the problem of deciding when an SPA (e.g., a specialized shape-from-X algorithm or an acoustic filter) with particular parameter settings is appropriate to a given environment may involve nonlinear control or be unsolvable with current control theory techniques.

It is important to clarify the relationship between the IPUS approach and the classic control theoretic approach [42]. Control theory uses stochastic-process concepts to characterize signals, and these characterizations are limited to probabilistic moments, usually no higher than second-order. Discrepancies between these stochastic characterizations and an SPA's output data are used to adapt future signal processing. In contrast, the IPUS architecture uses high-level symbolic descriptions (i.e., interpretation models of individual sources) as well as numeric relationships between the outputs of several different SPAs to characterize signal data. Discrepancies between these characterizations and SPAs' output data are used to adjust future signal processing. Classic adaptive control should therefore be viewed as a special case of an IPUS architecture, where the interpretation models are described solely in terms of probabilistic measures and low-level descriptions of signal parameters.

B.6 The IPUS Acoustic Interpretation Testbed

This section presents an acoustic interpretation testbed that we designed to experimentally examine the behavior of an IPUS-based system. The testbed runs on a TI Explorer II+ and is implemented in approximately 1400Kb of source code. All SPAs are implemented in software. Figure B.4.5b shows the IPUS architecture's realization in this testbed. The testbed description is divided into two parts. In the first part we describe how each of the generic IPUS components was instantiated in the testbed. The second part describes the testbed's acoustic domain knowledge as background for understanding the trace in Section B.7.2.

B.6.1 Instantiated IPUS Components

As we describe the testbed KSs, note that our KS algorithm descriptions are only intended as *instances* of algorithms that can implement the components. For example, the testbed's actual discrepancy diagnosis algorithm will be seen to be means-ends analysis using difference operators to encode the distortions implied by Fourier theory SPA processing models. Other algorithms using rules or case-based reasoning or qualitative models to apply the SPA processing models could have been used, as long as they provided the same diagnostic functionality.

Discrepancy Detection

The task of detecting discrepancies is distributed among all the knowledge sources responsible for interpreting correlates or lower-level interpretations as higher-level concepts. When executed, each such KS checks to see if any support is available for a higher-level concept. If none can be found, or if only partially supportive data is available, the KS will record this as a SOU (see Section B.4.3) in the problem solving model, to be resolved at the discretion of the focusing heuristics. At the end of each data block's numeric signal processing, a fault discrepancy detection KS is executed to check if SPA outputs are consistent with each other. Again, when discrepancies are found, SOUs are posted in the problem solving model. The basic SOU types defined in the RESUN framework are:

- **partial evidence** – Denotes the fact that there is incomplete evidence for the hypothesis.
- **possible alternative support** – Denotes the possibility that there may be alternative evidence that could play the same role as a current piece of support evidence.
- **possible alternative explanation** – Denotes the possibility that there may be alternative explanations for the hypothesis.
- **alternative extension** – Denotes the existence of competing, alternative versions of the same hypothesis.
- **negative evidence** – Denotes the failure to be able to produce some particular support evidence or to find any valid explanations.

In the integration of the IPUS and RESUN frameworks, an important issue is the relationship between the SOUs associated with various hypotheses and the discrepancy descriptions generated by the discrepancy detection process. Our architecture uses the following relationships:

1. **Conflict-type Discrepancies and SOU's.** Conflict-type discrepancies occur when signal processing output data does not match expectations. When an expectation is first posted, it has no supporting evidence because none has been searched for yet. To reflect this fact, the expectation is annotated with a PARTIAL SUPPORT SOU, which is a *partial evidence* type of SOU. To resolve this uncertainty, IPUS searches for evidence matching the expectations. If any portion of the expectation is unmatched after supporting evidence has been sought, a conflict discrepancy is raised for that expectation. When a conflict discrepancy is detected, a SUPPORT EXCLUSION SOU, a *negative evidence* type of SOU, is attached to the expectation.

2. **Fault-type Discrepancies and SOU's.** Fault-type discrepancies arise when two different signal processing algorithms produce conflicting hypotheses about the same underlying signal data. In such cases, a composite hypothesis is created that is a copy of the more reliable of the two data hypotheses and is considered to be an extension of that hypothesis. A link labeled with a *negative evidence* SOU (in particular, a SUPPORT LIMITATION SOU, which indicates that support for a hypothesis is limited until results of further processing are obtained) connects the less reliable hypothesis to the composite hypothesis.

3. **Violation-type Discrepancies and SOU's.** A violation-type discrepancy occurs when signal processing output data violates the *a-priori* known characteristics of the entire class of possible input signals in the application domain. When such an output data hypothesis is posted on the interpretation blackboard, a CONSTRAINT SOU, a *negative evidence* type of SOU, is attached to it. This SOU contains a description of the violated condition.

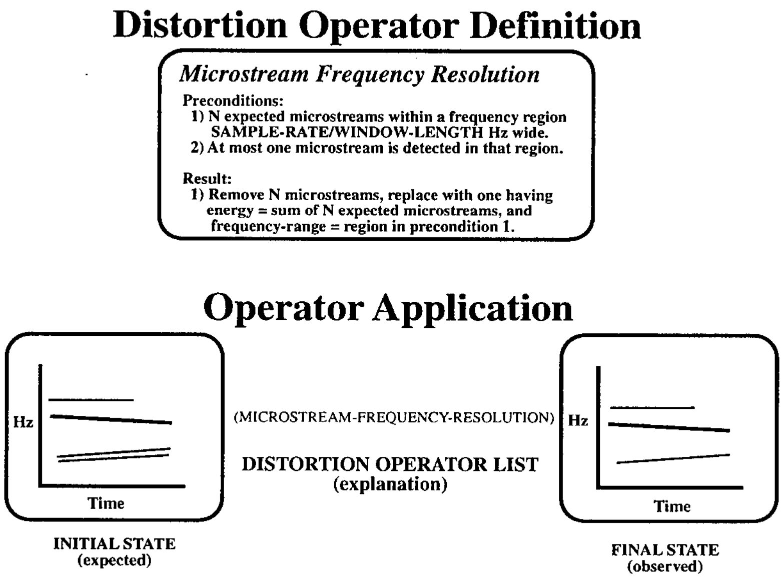
In addition to the discrepancy detection components of the interpretation KSs (that perform conflict discrepancy detection), the testbed contains KSs for fault discrepancy detection and violation discrepancy detection.

The actual comparisons implemented in the testbed discrepancy detection components were derived from an inspection of the SPAs available to the testbed designers and the context-dependent and context-independent features these SPAs' correlates could support.

Discrepancy Diagnosis

The discrepancy diagnosis KS is designed to take advantage of the fact that the SPA processing models from an environment's signal processing theory can predict how SPA output will be distorted if the SPA is misapplied. Refering back to a previous example, assume that an STFT with an analysis window of W sample points is applied to a signal sampled at R samples per second. If the signal came from a scenario containing frequency tracks closer than R/W Hz, Fourier theory predicts that the tracks will be merged in the STFT's computed correlates.

Our testbed instantiation of the diagnosis component models this knowledge in a database of formal distortion operators. When applied to an abstract description of anticipated or computed correlates, an operator returns the description modified to contain the operator's distortion. The KS uses these operators in a means-ends analysis framework incorporating multiple abstraction levels and a verification phase [37] to "explain" fault, violation, and conflict discrepancies. The KS takes two inputs: an *initial state* representing anticipated correlates and a *goal state* representing the computed correlates. The formal task of the KS is to generate a distortion operator sequence mapping the initial state description onto the goal state description. Figure B.6.6 illustrates the formal operator definition of the previously described frequency resolution distortion that the STFT SPA data correlates can be subject to, as well as its use in a short explanation.



Distortion Operator Definition

Microstream Frequency Resolution

Preconditions:
1) N expected microstreams within a frequency region SAMPLE-RATE/WINDOW-LENGTH Hz wide.
2) At most one microstream is detected in that region.

Result:
1) Remove N microstreams, replace with one having energy = sum of N expected microstreams, and frequency-range = region in precondition 1.

Operator Application

Figure B.6.6: *Microstream Frequency Resolution Operator from the Acoustic Interpretation Testbed. When applied to a state, the operator replaces each set of expected microstreams whose members are closer than SAMPLE-RATE/WINDOW-LENGTH with a single microstream, reflecting the resolving limits associated with the current value of WINDOW-LENGTH. In the short example illustrated, this operator effectively reduces the differences between the expected state and the observed state.*

The KS's search for an explanatory distortion operator sequence is iteratively carried out using progressively more complex abstractions of the initial and goal states, until a level is reached where a sequence can be generated using no more signal information than is available at that level. Thus, the KS mimics expert diagnostic reasoning in that it offers simplest (shortest) explanations first [41]. Once a sequence is found, the KS enters its verify phase, "drops" to the lowest abstraction level, and checks that each operator's pre- and post-conditions are met when all available state information is considered. If verification succeeds, the operator sequence and a diagnosis region indicating the signal hypotheses involved in the discrepancy are returned. If it fails, the KS attempts to "patch" the sequence by finding operator subsequences that eliminate the unmet conditions and inserting them in the original sequence. If no patch is possible, and no alternative explanations can be generated, the involved signal hypotheses are annotated with an SOU with a very negative rating. Figure B.6.7 outlines the plan-and-verify strategy of the diagnostic process.

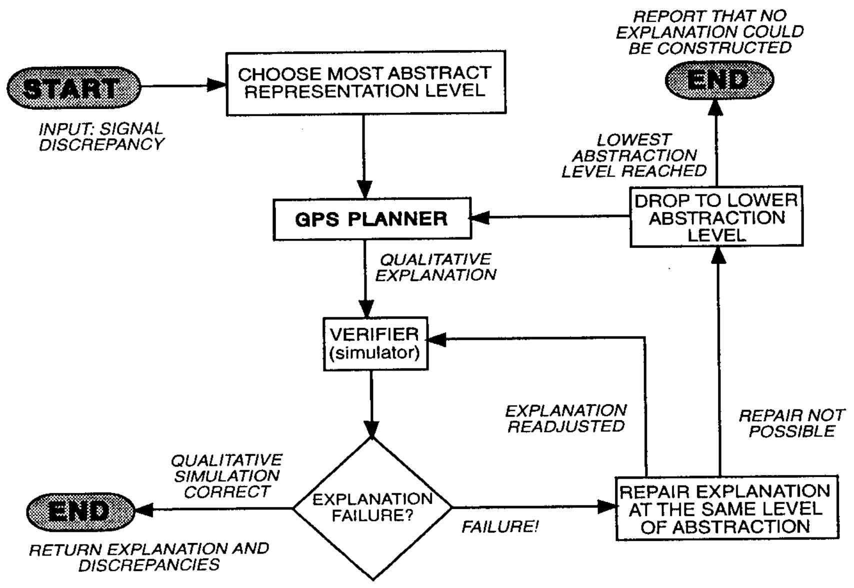

REPORT THAT NO EXPLANATION COULD BE CONSTRUCTED
START
CHOOSE MOST ABSTRACT REPRESENTATION LEVEL
END
INPUT: SIGNAL DISCREPANCY
LOWEST ABSTRACTION LEVEL REACHED
GPS PLANNER
DROP TO LOWER ABSTRACTION LEVEL
QUALITATIVE EXPLANATION
VERIFIER (simulator)
EXPLANATION READJUSTED
REPAIR NOT POSSIBLE
QUALITATIVE SIMULATION CORRECT
END
EXPLANATION FAILURE?
FAILURE!
REPAIR EXPLANATION AT THE SAME LEVEL OF ABSTRACTION
RETURN EXPLANATION AND DISCREPANCIES

Figure B.6.7: *The plan-and-verify strategy of the IPUS discrepancy diagnosis knowledge source.*

An issue not addressed in earlier work [37] that arose in the development of IPUS is the problem of inapplicable explanations. Sometimes the first explanation offered by the KS will not enable the reprocessing mechanism to eliminate a discrepancy. In these cases, the architecture's control framework (expressed as control plans) permits reactivation of the diagnostic KS with the previous explanation supplied as one that must not be returned again. To avoid repetition of the search performed for the previous explanation, the KS stores with its explanations the search-tree context it was in when the explanation was produced. The KS's search for a new explanation begins from that point.

The discrepancy diagnosis KS's output is also used to modify expectations for how future support evidence should appear under the current parameter settings. Each distortion operator contains a logical "support specification" of how data that is expected can appear distorted when processing parameters take on the current parameter values. When a complete distortion-operator sequence is generated, all operators' support-specifications are conjunctively combined to form a single expectation specification. This specification is then attached to the expectation units of the hypotheses involved in the original discrepancy. For those feature hypotheses, this annotation reduces the quality-level required for future evidence. The specification indicates to the system that when it is seeking data correlates from an SPA X for object features which were previously distorted by X, it can use data correlates which match the specification's distortions *without* raising a discrepancy.

Signal Reprocessing

Once the distortions have been hypothesized by the discrepancy diagnostic reasoning process, the next task is to search for the appropriate SPAs and control parameter settings under which signal reprocessing may remove those distortions. Figure B.6.8 illustrates the organization of the reprocessing knowledge source used in the testbed. This reprocessing portion of the architecture consists of the following major components: *situation assessment, reprocessing-plan selection, and reprocessing-plan execution.* The input to the reprocessing knowledge source includes a description of the input and output signal states (see diagnostic reasoning section above), the distortion operator sequence hypothesized by the diagnosis stage, and a description of the discrepancies present between the input and output signal states. The situation assessment phase uses case-based reasoning to generate multiple reprocessing plans, each of which has the potential of eliminating the hypothesized distortions present in the current situation. Plans for eliminating various categories of distortions are stored in a knowledge base. Figure B.6.9 shows the definition for one reprocessing plan schema from our acoustic interpretation testbed. This reprocessing plan's role is to extract a short high-energy contour which was missed by the front-end STFT instance but whose presence was indicated by the front-end's time-domain energy tracker.

From the retrieved set of applicable plans, one is selected during the plan-selection stage. Selections are governed by "cost" criteria such as plan execution time. The execution of a reprocessing plan consists of incrementally adjusting the SPA control parameters, applying the SPA to the portion of the signal data that is hypothesized to contain distortions, and testing for discrepancy removal. The incremental process is necessary because the situation description is often at least partially qualitative, and therefore it is generally impossible to predict exact values for the control parameters to be used in the reprocessing.

Reprocessing continues until the goal of distortion removal is achieved or it is concluded that the reprocessing plan has failed. Currently there are two independent criteria for determining plan failure in IPUS. The first criterion simply considers the number of plan iterations. If the number surpasses a fixed threshold, failure is indicated automatically. The second criterion relies on fixed lower and upper bounds

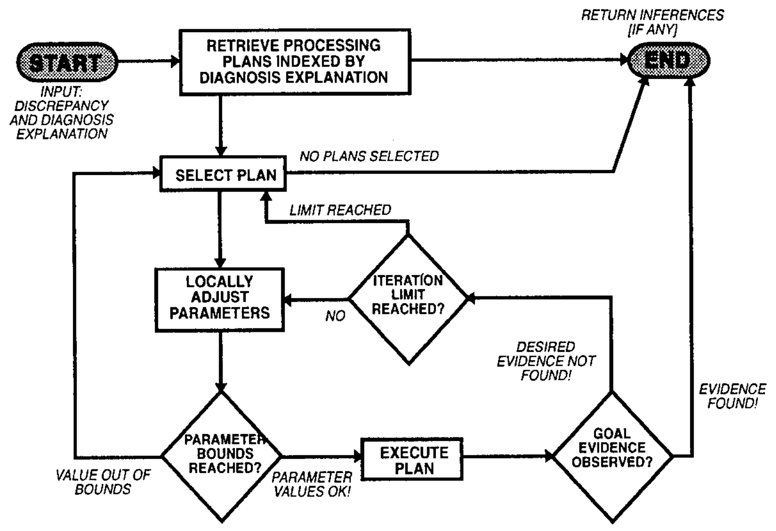

START
INPUT: DISCREPANCY AND DIAGNOSIS EXPLANATION
RETRIEVE PROCESSING PLANS INDEXED BY DIAGNOSIS EXPLANATION
RETURN INFERENCES [IF ANY]
END
SELECT PLAN
NO PLANS SELECTED
LIMIT REACHED
LOCALLY ADJUST PARAMETERS
NO
ITERATION LIMIT REACHED?
DESIRED EVIDENCE NOT FOUND!
EVIDENCE FOUND!
PARAMETER BOUNDS REACHED?
VALUE OUT OF BOUNDS
PARAMETER VALUES OK!
EXECUTE PLAN
GOAL EVIDENCE OBSERVED?

Figure B.6.8: *The IPUS reprocessing knowledge source's framework*

for signal processing parameters. If a plan reiteration requires a parameter value outside of its prespecified range, the plan is considered to have failed.

When failure is indicated, the discrepancy diagnosis process can be re-invoked to produce an alternative explanation for the distortions present in the original signal data. If no alternative explanation is available (i.e., the diagnostic knowledge source fails to find another distortion operator sequence), an IPUS-based system annotates the hypothesized features involved in the discrepancy with SOUs indicating low confidence due to unresolvable discrepancies. These SOUs' effects on the features' confidence levels are then propagated to object interpretations based on those features, causing their existence to be disbelieved more strongly.

Differential Diagnosis

In the course of processing signal data, IPUS-based systems will encounter signals that could support several alternative interpretations. We include the differential diagnosis KS to produce reprocessing plans that will enable the system to prune the interpretation search space when ambiguous data correlates are encountered. Its input is the ambiguous data's set of alternative interpretations, and its output is a triple containing:

1. the time region in the signal data to be reprocessed
2. the support evidence (verification goals) that must be found for each interpretation
3. the set of reprocessing plans and parameter values proposed for revealing the desired support evidence.

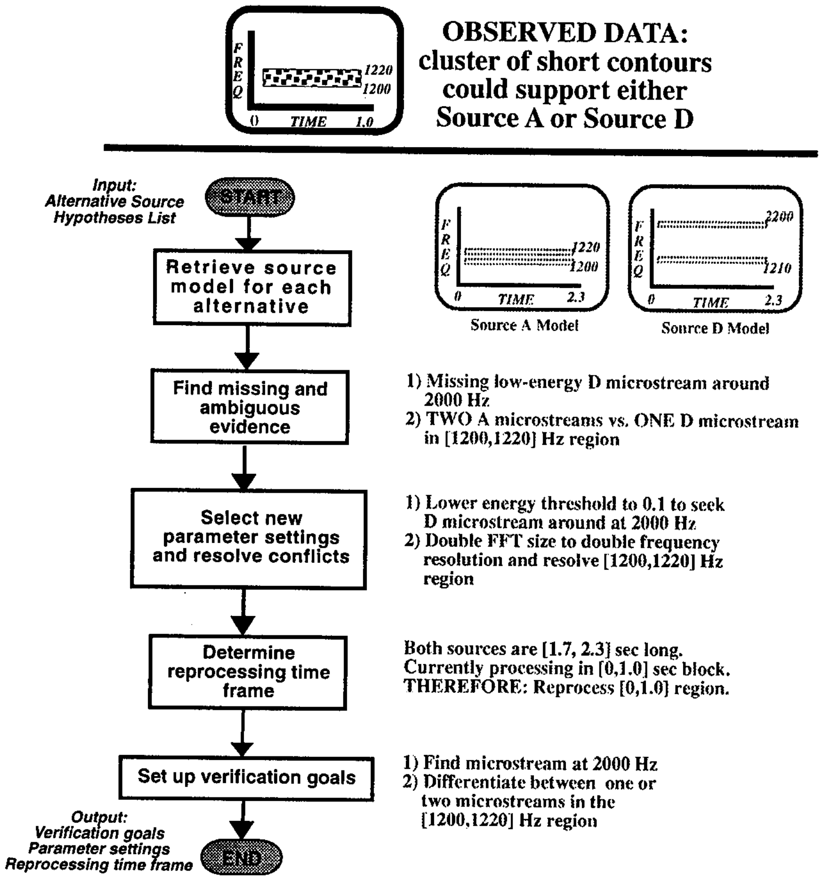
```
(CONTOUR-1
  (state :name faulty
         :hyp-type contour
         :hyp =x)
  (state :name faultless
         :hyp-type contour
         :hyp =y)
  (operator-sequence (STFT-TIME-RESOLUTION))
  (discrepancy
         :type fault
         :name MISSING-STFT-CONTOUR-PRESENT-TD-CONTOUR
         :level contour
         :duration =x1
         :energy =x2
         :frequency =x3
         :expected-region =z)
  --> (reprocessing-plans
         ((reprocessing-plan
             :input-variables (:faulty-hyp =x
                               :faultless-hyp =y
                               :expected-region =z)
             :parameters (*STFT-OVERLAP*
                          *WINDOW-LENGTH*
                          *STFT-PEAK-ENERGY-THRESHOLD*)
             :parameter-changes
               ((lambda (p) (/ p 8))
                (lambda (p) (/ p 4))
                (lambda (p) 0.9))
             :primitive-plans (delete-all-reprocessing-units
                               reprocess-spectra-for-contours
                               reprocess-contours)
             :goal-condition (contours-present?)))))
```

Figure B.6.9: *The definition for a reprocessing plan from the acoustic interpretation testbed to handle the distortion-operator sequence (CONTOUR-TIME-RESOLUTION). The plan specifies that on each iteration of the primitive plan list, the STFT-OVERLAP and WINDOW-LENGTH parameter values are divided by 8 and 4, respectively, while the STFT-PEAK-ENERGY-THRESHOLD parameter value is maintained at 0.9. At the end of each iteration, the goal-condition CONTOURS-PRESENT? is tested for. This goal requires that the sought high-energy contour appear.*

Our implementation of this KS uses the following strategy. The KS first compares the interpretation hypotheses to determine their overlapping regions. Any observed evidence in these regions is labeled "ambiguous". The KS then determines the hypotheses' discriminating regions (e.g., `Hyp1`, and no other hypothesis, has a microstream at 2000 Hz). For each discriminating region where no evidence was observed, the KS posits an explanation for how the evidence could have gone undetected, assuming the hypothesized source was actually present. Using these explanations as indices into a plan database, the KS retrieves reprocessing plans and parameter values that should cause the missing evidence to appear. At this point the ambiguous evidence is considered. The KS seeks for multiple signal structures within each overlapping region (e.g., a region that contains data that could support one microstream of a hypothesis or two microstreams of another hypothesis), and selects processing plans to produce data with better structural resolution in the regions of overlap.

If the missing-evidence processing plan set and the ambiguous-evidence plan set intersect, the intersection forms the third element of the output triple. If the intersection is empty, the missing-evidence plan set forms the third element of the output triple. Finally, if the missing-evidence plan set is empty, the ambiguous-evidence plan set is returned. The rationale behind this hierarchy of plan set preference is that this ordering will return the most likely plans for producing evidence that could eliminate interpretations from further consideration. The region of mutual temporal overlap for the alternative hypotheses defines the reprocessing time region in the output triple, and the ambiguous and missing data that is handled by the reprocessing plan set defines the support evidence in the output triple. The output triple's reprocessing plan is then executed as in the reprocessing KS until either the parameter-value limits are exceeded or at least one of the pieces in the support evidence set is found after a reprocessing. Figure B.6.10 depicts a typical execution for the testbed differential diagnosis KS.

We should note that the explanatory reasoning performed in the differential diagnosis KS for missing evidence is primitive compared to that available in the discrepancy diagnosis KS; there is not a rich set of explanations available. Only simple single-operator distortions like loss of low-energy components due to energy thresholding are considered. This design is justified because the differential diagnosis KS's role is to trigger reprocessing that quickly prunes large areas of underconstrained interpretation spaces, *without preference* for any particular interpretation. On the basis of this specification, it is not appropriate to devote time consuming, sophisticated reasoning to the generation of missing-evidence explanations. For related reasons, the differential diagnosis KS does not return support specifications that reduce the quality-level required for future evidence. The KS's shallow explanations generated for finding contrasts within a set of several sources might not justify the acceptance of lower quality evidence for a single source from that set.



FREQ 1220 1200 0 TIME 1.0
OBSERVED DATA: cluster of short contours could support either Source A or Source D
Input: Alternative Source Hypotheses List
START
Retrieve source model for each alternative
FREQ 1220 1200 0 TIME 2.3
Source A Model
FREQ 2200 1210 0 TIME 2.3
Source D Model
Find missing and ambiguous evidence
1) Missing low-energy D microstream around 2000 Hz
2) TWO A microstreams vs. ONE D microstream in [1200,1220] Hz region
Select new parameter settings and resolve conflicts
1) Lower energy threshold to 0.1 to seek D microstream around at 2000 Hz
2) Double FFT size to double frequency resolution and resolve [1200,1220] Hz region
Determine reprocessing time frame
Both sources are [1.7, 2.3] sec long.
Currently processing in [0,1.0] sec block.
THEREFORE: Reprocess [0,1.0] region.
Set up verification goals
1) Find microstream at 2000 Hz
2) Differentiate between one or two microstreams in the [1200,1220] Hz region
Output: Verification goals Parameter settings Reprocessing time frame
END

Figure B.6.10: *A flowchart for the IPUS differential diagnosis KS and its execution in a typical acoustic scenario. In this example a database query returns more than one sound model whose frequency components overlap the observed data in the* [1200, 1220] *Hz region. For each model, the IPUS system posts an interpretation hypothesis supported by the observed data. In the problem-solving model, an ALTERNATIVE-EXPLANATION SOU is recorded for each hypothesis. These SOUs are left unresolved until selected by the system's focusing heuristics.*

In cases where an IPUS system prefers a particular interpretation over alternatives, and needs an explanation for why the interpretation is missing certain support, it will make use of the discrepancy diagnosis KS, with the initial state reflecting the preferred interpretation.

B.6.2 Testbed Domain Knowledge

The testbed consists of a blackboard with eight evidence abstraction levels, KSs for the primary IPUS components and for inferring hypotheses between different abstraction levels, an acoustic source library, and control plans. The testbed version described in this paper is called configuration *C.1*.[3]

Figure B.6.11 describe the information represented in the evidence abstractions. At the lowest level are waveform segments derived from the input waveform. Each segment is a collection of points to which some SPA will be applied. Time-domain statistics such as zero-crossing density, average energy, etc, are also maintained for segments. The second level consists of spectral hypotheses derived for each waveform segment through Fourier-Transform-based algorithms such as the STFT and Wigner-Distribution [13] algorithms. The third level consists of peak hypotheses derived for each spectrum and is used to support narrow-band features of sounds. The fourth level consists of contour hypotheses, each of which corresponds to a group of peaks whose time indices, frequencies, and amplitudes represent a contour in the time-frequency-energy space with uniform frequency and energy behavior. The fifth level contains microstream hypotheses supported by one contour or a sequence of contours. Each microstream has an energy pattern consisting of an attack region (signal onset), a steady region, and a decay (signal fadeout) region. In the sixth level we represent noisebeds as wideband frequency regions supported by regions within spectra. Noisebeds represent the wideband component of a sound source's acoustic signature. Usually microstreams form "ridges" on top of noisebed "plateaux," but not every noisebed has an associated microstream. Groups of microstreams and noisebeds synchronized according to time and/or other psychoacoustic criteria such as harmonic frequency sets support stream hypotheses in the seventh level. Bregman [7] provides a highly detailed account of various psychoacoustic streaming processes. At the eighth level, sequences of stream hypotheses are interpreted as sound-source hypotheses.

Sources are represented in the source database by an acoustic grammar specifying microstream and noisebed frequency ranges and permissible ranges of energy relationships among microstreams and noisebeds within source streams. The grammar also specifies the permissible range of durations for each source's microstreams and streams, and the stream sequences and periodic patterns that characterize the source.

B.7 Acoustic Interpretation Testbed Operation

In this section we provide a detailed analysis of the acoustic interpretation testbed's behavior as it interprets the waveform data from an acoustic scenario constructed

[3]Configuration *C.2* is currently under development as a platform for exploring approximate processing and scaling issues.

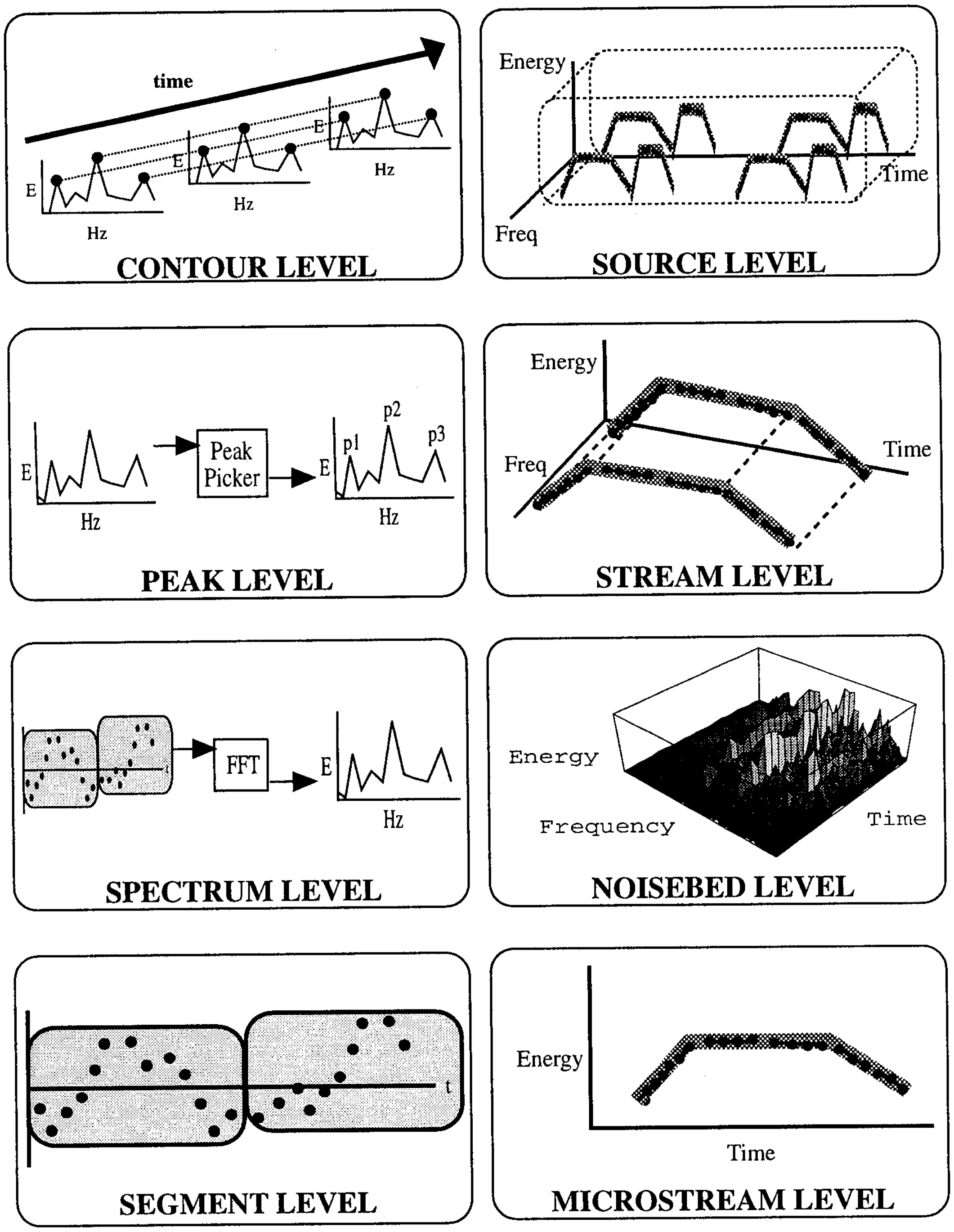


Figure B.6.11: *Testbed Evidence Abstractions.*

from real-world, narrowband signals. By showing the IPUS components' functionality and their use of formal relationships between signal characteristics and SPA parameters, the example illustrates the important role that a formal theory of signal processing can play in signal interpretation.

B.7.1 Scenario Overview

Figure B.7.12a shows the time-domain waveform (sampled at 8KHz) provided to the testbed, while Figure B.7.12b shows how the sources in the scenario would appear using context-appropriate processing. `Phone-Ring` and `Siren-Chirp` are 1.2 times as energetic as `Buzzer-Alarm`, and `Glass-Clink` is an impulsive source 3.0 times as energetic as `Buzzer-Alarm`. Figure B.7.12c shows how the events are distorted when the testbed's initial front-end configuration is applied throughout the scenario.

The testbed was initially configured to interpret waveform data in 1.0-second blocks, and to identify quickly any occurrences of `Siren-Chirp`. In particular, the system's SPA parameters were set to detect `Siren-Chirp`'s steady-energy behavior:

FFT-SIZE: 512
The number of uniformly-spaced frequency samples computed for each Short-Time Fourier Transform (STFT) analysis window position.

WINDOW-LENGTH: 512
The number of data points to which each FFT in the STFT algorithm is applied ($\leq$ FFT-SIZE).

DECIMATION: 512
The number of points between consecutive STFT analysis window positions. The value was set to 512 to permit the fastest possible processing of the data.

PEAK-THRESHOLD: 0.09
Spectrum points with energy below this value are rejected by the peak-picking algorithm.

For processing this example, the testbed's source database was loaded with models for the five narrowband sources shown in Figure B.7.13. In the figure the sources' frequency components are labelled by single-frequency values only for clarity; the formal source definitions have frequency *ranges* specified for each component.

There are several critical actions that the IPUS acoustic testbed must perform if it is to reasonably analyze Figure B.7.12a's signal. In block 1, the testbed encounters two alternative interpretations of the data in the $[420, 500]$ frequency region. That is, there is the possibility that it could be caused by `Phone-Ring` or `Car-Horn`, or even both occurring simultaneously. One reason for this confusion stems from the fact that the energy threshold setting for the peak-picking algorithm is high and would prevent `Car-Horn`'s low-energy microstream from being detected if in fact it

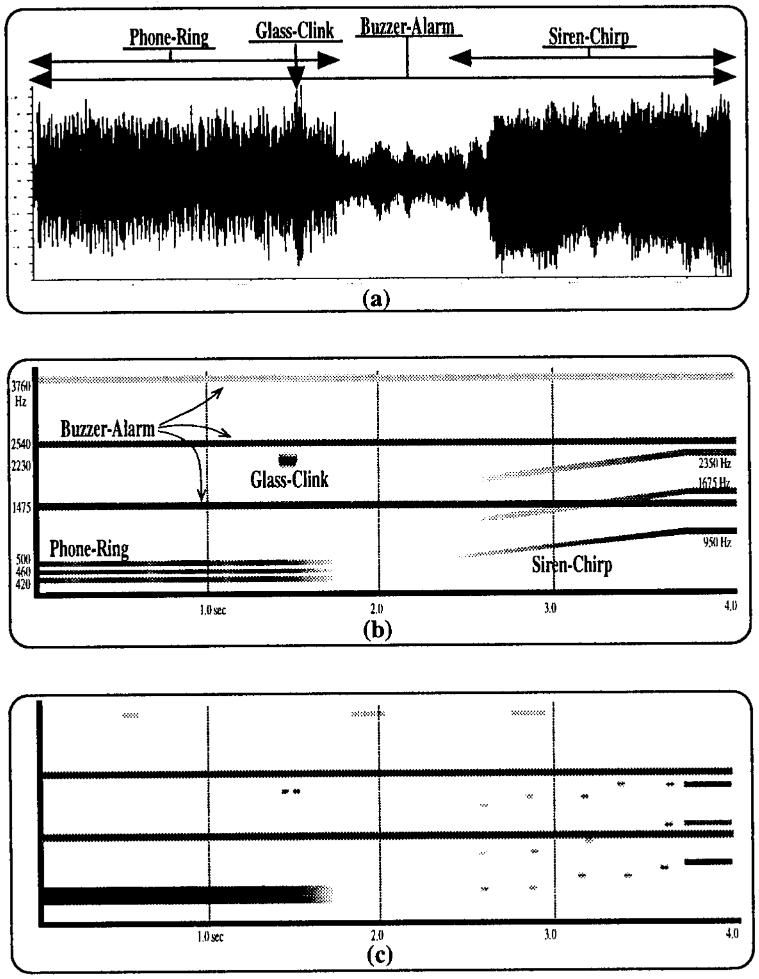


Figure B.7.12: *Acoustic Scenario Events. Figure B.7.12a shows the scenario's time-domain waveform. Figures B.7.12b and B.7.12c show the scenario's frequency-domain events. Darker shading indicates higher frequency-domain energy.*

were present. The second reason is that the frequency-sampling provided by the STFT algorithm's FFT-SIZE parameter does not provide enough frequency sample points to resolve the [420, 500] region into `Phone-Ring`'s three microstreams. The uncertainty in this situation is resolved through reprocessing under the direction of differential diagnostic reasoning, which increases resolution and decreases the energy threshold.

During block 1's analysis, the testbed also determines that `Buzzer-Alarm`'s track at 3760 Hz is missing. One reason for this is that the track's energy might be too low for the peak-picker's PEAK-THRESHOLD parameter setting. The discrepancy is resolved through reprocessing the previously-produced spectra with a lower PEAK-THRESHOLD value.

In block 2, the testbed detects a discrepancy between the outputs of its time-

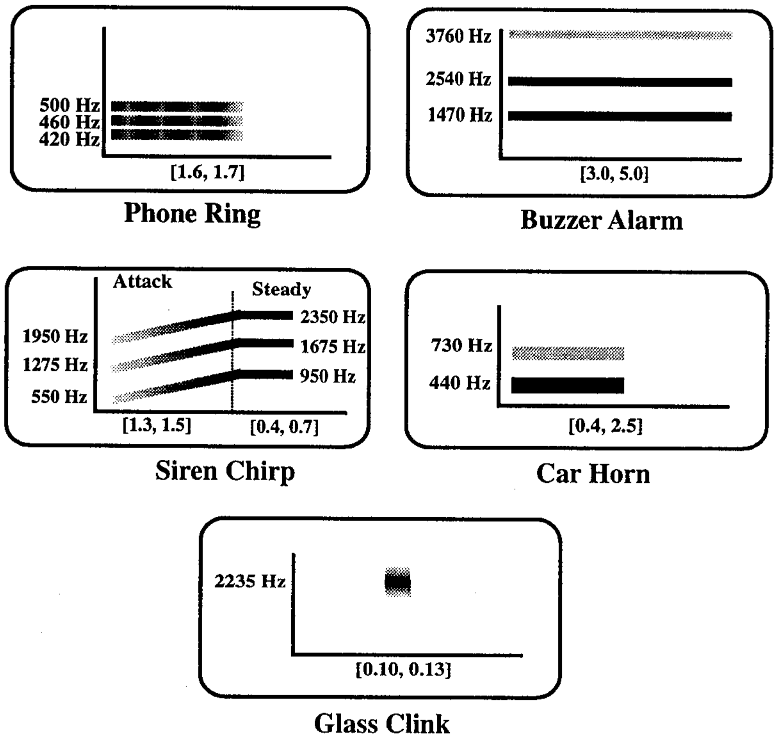


Figure B.7.13: *IPUS Source Database. The vertical axis represents frequency and the horizontal axis represents time in seconds. The energy changes for each microstream are represented qualitatively by the shading gradations. Note that* `Phone-Ring` *is a ring from a phone different from the one in Figure B.2.3.*

domain energy estimator SPA and its STFT SPA. The energy estimator SPA detects a substantial energy increase followed about 0.1 seconds later by a precipitous decrease. The STFT SPA, however, produces no significant set of peaks to account for the signal energy flux. This is because the algorithm's decimation parameter is too high. The testbed also detects a discrepancy between expectations established from block 1 for the [420, 500] frequency region and the STFT SPA's output. The STFT SPA produces short contours that cannot support the expected microstreams for `Phone-Ring` because of inadequate frequency sampling in the region. Both discrepancies are resolved by reprocessing. The first discrepancy is resolved through reprocessing with a smaller DECIMATION value and smaller STFT intervals, while the second is resolved through reprocessing with the finer frequency sampling provided by a 1024 FFT-SIZE.

In block 3, `Siren-Chirp`'s attack interacts with the poor time-resolution of the STFT SPA to produce a set of widely-separated short contours that the testbed cannot immediately interpret as the attack portion of microstreams. In block 4, however, the testbed uses the discovery of `Siren-Chirp`'s steady region as the basis for re-interpreting block 3's short contours as evidence for the sound's attack region.

B.7.2 Testbed Trace

The following is a high-level trace of the significant events that occurred as the system processed the signal in Figure B.7.12a.

BLOCK 1

- **Bottom-Up Processing:** The testbed focusing heuristics specify that spectral information be gathered for the input waveform sampled during block 1. It is processed by a KS representing the STFT signal processing algorithm and a KS that uses a time-domain algorithm for estimating waveform energy as a function of time. Continuing in a data-driven manner, the spectra peaks produced are grouped by similar frequency and energy into contours.

- **Seek Evidence for Current Expectations:** The focusing heuristics next direct the testbed to act upon current high-level expectations and search for support evidence. In deciding what evidence to examine first, the heuristics choose to look for any evidence in the steady-phase frequency regions of high-priority sources (`Siren-Chirp` in this case). No contours are found in these regions. At this point in the experiment, there are no other explicit source expectations.

- **Drive Unexplained Data to Higher Levels:** Contours in the $[1460, 1480]$ and $[2530, 2550]$ Hz regions are used to support microstream hypotheses. These in turn are used to support a `Buzzer-Alarm` source hypothesis. However, support for `Buzzer-Alarm`'s third microstream is not found in the peak-picker's correlates, causing a conflict discrepancy SOU to be posted with the source.

- **Discrepancy-Detection:** The testbed uses the heuristic that short contours[4] should not be used as microstream evidence. Because the block has a large number of short contours relative to the total number of contours detected, the testbed performs discrepancy detection to determine if there are tight short-contour clusters that could indicate distorted sources. The system finds such a cluster in the $[420, 500]$ Hz range, and then queries the source database to find a source hypothesis to explain the cluster. `Phone-Ring` and `Car-Horn` are retrieved because at least one of each source's frequency components overlaps the cluster. Therefore the testbed posts both sources as alternative explanations for the contour cluster. This use of short contours in place of long contours to support interpretations raises a violation discrepancy, since the *a priori* expectation that sources are indicated only by long contours is violated.

[4]Contours having between 1 and 3 peaks. Short contours could be the result of random noise, and the system should apply as little computing time as necessary to the processing of noise.

- **Handle Selected Uncertainties:** At this point four SOUs have been posted: one each for the violation discrepancies associated with `Phone-Ring` and `Car-Horn` being supported by a cluster, one for the uncertainty associated with the existence of competing interpretations for the same cluster, and one for `Buzzer-Alarm`'s missing microstream. The focusing heuristics elect to resolve the uncertainty associated with the alternative explanations. For doing this, the control plans specify a strategy of first performing differential diagnosis and using its results to guide data reprocessing.

  1. **Differential Diagnosis:** The differential diagnosis KS determines features of the two sources that should be searched for in the signal data because their presence or absence will permit differentiation between the alternatives. In this case the KS selects the low-energy, 900 Hz microstream of `Car-Horn` and the number of microstreams in the $[420, 500]$ Hz region for each source (`Phone-Ring` has 3, `Car-Horn` has 1) as discriminating features. It specifies that a lower energy-threshold be used to attempt to "bring out" `Car-Horn`'s low-energy microstream at 730 Hz. To attempt to find `Phone-Ring`'s three microstreams, it specifies an FFT-SIZE value of 1024 to increase the frequency sampling in the $[420, 500]$ Hz region. Note that the testbed at this time is not committed to either interpretation, nor to the possibility that **both** sources are present. Any decisions will wait for the results of reprocessing.
  2. **Differential Reprocessing:** The reprocessing KS is executed and the sought-after `Car-Horn` microstreams are not found. However, *three* well-defined contours are found in the $[420, 500]$ Hz range that can support `Phone-Ring`'s microstreams. Therefore `Phone-Ring`'s belief is increased, while `Car-Horn`'s belief is decreased. `Car-Horn`'s belief level is very low at this point and is no longer considered as a significant alternative explanation for the original stream hypothesis. Note that this reprocessing opportunistically resolves not only the competing-interpretation uncertainty, but also `Phone-Ring`'s violation-discrepancy uncertainty.

- **Handle Selected Uncertainties: (continued)** Focusing heuristics now select the conflict discrepancy SOU of `Buzzer-Alarm`'s missing microstream for resolution. This is handled through calling the discrepancy diagnosis KS and executing a reprocessing plan based on its explanation.

  1. **Discrepancy Diagnosis:** The diagnosis KS produces the explanation (MS-ENERGY-THRESHOLDING) for the discrepancy. That is, peak-picker SPA's PEAK-THRESHOLD parameter has a value too high to detect enough peaks to generate long contours for the microstream.

2. **Discrepancy Reprocessing:** The reprocessing KS uses the explanation to decide to reprocess spectra from the entire block with a peak-picker SPA having a reduced PEAK-THRESHOLD value of 0.04. This produces seven peaks in the [3750, 3770] Hz region, which create a significant-length contour. This contour's existence resolves the conflict discrepancy. `Buzzer-Alarm`'s 3760 Hz microstream is annotated with a support specification that indicates that very short (one peak) contours or none at all are acceptable evidence as long as the PEAK-THRESHOLD value is higher than 0.04.

- **Define Expectations:** Because `Phone-Ring`'s description indicates that its steady region is approximately 1.7 seconds long, and at most 1.0 second has been found, an explicit expectation for `Phone-Ring`'s microstreams is posted for block 2's time period. Explicit expectations for the continuation of `Buzzer-Alarm`'s microstreams are also posted for block 2.

BLOCK 2

- **Bottom-Up Processing:** Bottom-up processing creates spectra and contours for block 2. `Glass-Clink` emits a high-energy, short-duration (0.12 sec) signal burst. The time-domain algorithm detects a sharp increase followed by a sharp decrease in signal energy, whereas the STFT produced no peaks to generate a significant-length contour that started and stopped around the times indicated by the signal-energy shifts. The testbed control plans were designed to perform fault discrepancy detection immediately after bottom-up signal processing is completed. This causes a fault discrepancy to be detected between the time-domain energy monitoring algorithm and the STFT algorithm.

- **Seek Evidence for Current Expectations:** Since the duration of the fault discrepancy indicates that it is not related to `Siren-Chirp`,[5] the focusing heuristics act on `Siren-Chirp`'s priority and decide to examine data found in the source's expected frequency regions. No contours are found in these regions.

- **Handling Selected Uncertainties:** The testbed's focusing heuristics select fault-type SOUs for resolution before the control plans apply any interpretation KSs that might handle frequency regions affected by fault discrepancies. Thus, before the components of any non-priority expected sources are searched for, the fault discrepancy is selected for handling by the focusing heuristics. For this SOU, the control plans specify a strategy that executes discrepancy diagnosis followed by reprocessing.

[5]`Siren-Chirp`'s duration is much longer than the fault's.

1. **Discrepancy Diagnosis:** The diagnosis KS explanation for the fault discrepancy is (CONTOUR-TIME-RESOLUTION). That is, the STFT decimation is too high to detect enough peaks to generate contours of significant length to account for the signal energy increase.
2. **Discrepancy Reprocessing:** The reprocessing KS uses the explanation to decide to reprocess data from the 0.09-second time region (*not* the entire block) with an STFT SPA having a 256-point WINDOW-LENGTH, a 512-point FFT-SIZE, and a 192-point DECIMATION. This produces four peaks in the $[2230, 2240]$ Hz region, which create a significant-length contour. This contour's existence resolves the fault discrepancy.

- **Seek Evidence for Current Expectations:** At this point, the focusing heuristics decide to gather evidence for explicit source expectations. Contours found in the expected regions of `Buzzer-Alarm` support that source's persistence into block 2. Note that when support for a source's microstreams is found, it is immediately propagated through the higher evidence levels (microstream and stream) to the source level. As happened in block 1, the front-end processing parameters produce a cluster of short contours in the $[420, 500]$ Hz range. The testbed's short-contour heuristic leads to a lack of support for the persistence of `Phone-Ring`'s microstreams into block 2.

- **Discrepancy Detection:** The testbed checks for conflict and violation discrepancies. The lack of support for `Phone-Ring`'s microstreams raises a conflict discrepancy.[6] No violation discrepancies are found.

- **Handle Selected Uncertainties: (continued)** The focusing heuristics select the conflict SOU in `Phone-Ring`'s three microstreams for resolution. Control plans specify a strategy of discrepancy diagnosis followed by reprocessing.

  1. **Discrepancy Diagnosis:** The discrepancy detection KS returns the explanation (COARSE-FREQUENCY-SAMPLING); the STFT analysis was done with inadequate frequency sampling, causing the three microstreams to appear as the contour cluster actually observed. The KS also returns a support specification that in the next block under the same initial parameter settings, `Phone-Ring`'s microstreams will appear like the contour cluster again. In this scenario the support specification will not be useful, however, since based on `Phone-Ring`'s maximum possible duration the microstreams should not extend into block 3.[7]

[6]Remember that differential diagnosis does not annotate hypotheses with support specifications (see section B.6.1). Thus, `Phone-Ring`'s microstreams do not have specifications to prevent the testbed from registering the contour cluster as a distortion.

[7]A shortcoming of configuration *C.1* is that support specifications are not propagated across periodic streams. Thus, the support specification will not even be useful for any future rings. We are correcting this problem in configuration *C.2*.

2. **Discrepancy Reprocessing:** The reprocessing KS acts upon the diagnosis explanation and retrieves a processing plan directing that the data be reprocessed up to the microstream level of abstraction with an FFT-SIZE value of twice the original ($2 * 512 = 1024$ in this case). The doubling of FFT-SIZE provides finer frequency sampling in the spectra produced by the STFT algorithm. After one iteration of this plan, the desired microstreams are found, and their expectations in the next block are annotated with the discrepancy diagnosis KS's scenario-specification.

- **Drive Unexplained Data to Higher Levels:** The 0.12-second contour is found to match `Glass-Clink`'s characteristics. A hypothesis for the source is therefore posted.

- **Define Expectations:** Because `Buzzer-Alarm`'s model indicates that its steady behavior could continue for 3 to 28 more seconds, an explicit expectation for its continuation is posted for Block 3's time period. No expectation for `Phone-Ring` is posted because its model specifies a maximum duration of 1.7 seconds.

BLOCK 3

- **Bottom-Up Processing:** Block 3's signal data is now processed. Bottom-up processing culminates in the creation of contours.

- **Seek Evidence for Current Expectations:** `Siren-Chirp`'s frequency regions are examined for contours. Some short contours are present in this block from the source's attack phase, but because the testbed first recognizes sources by steady characteristics (due to their more predictable behavior), their presence does not cause the creation of a `Siren-Chirp` source hypothesis. Contours extending source `Buzzer-Alarm`'s microstreams are sought for and found.

- **Drive Unexplained Data to Higher Levels:** Because of their short lengths, the contours caused by `Siren-Chirp`'s attack phase are not selected to hypothesize the existence of any microstreams. They are simply labeled as possible-noise data. These contours are spread across a wide frequency region. Therefore, the violation-detection clustering algorithm does not find any high-density cluster to justify raising a discrepancy.

- **Define Expectations:** An expectation for `Buzzer-Alarm`'s microstreams to continue into block 4 is posted.

BLOCK 4

- **Bottom-Up Processing:** Block 4's signal data is now processed. Bottom-up processing culminates in the creation of contours.
- **Seek Evidence for Current Expectations:** The testbed first searches Siren-Chirp's frequency regions for contours. Contours supporting the source's steady region are detected, and a source hypothesis is posted. The testbed also finds contours to support Buzzer-Alarm's microstreams.
- **Handle Selected Uncertainties:** Because its attack region is unsupported, Siren-Chirp's confidence level is low. Due to Siren-Chirp's priority, the focusing heuristics decide to resolve this missing-support SOU. The control plans specify a strategy of accepting sets of short contours that reflect the slope of the chirp when grouping peaks into contours. No diagnosis is performed; the reprocessing is simply a context-dependent interpretation strategy for detecting chirps when their presence is suspected.

  1. **Reprocessing:** To find "enough" (60% in this case) of Siren-Chirp's attack region, the testbed must search back into block 3 and reinterpret the previously-detected but unrecognized short contours as valid attack-region contours. Siren-Chirp's attack region and its chirp characteristics are identified in the previous block's signal data[8]. At this point Siren-Chirp is determined to be present with high confidence.

At the end of the scenario the testbed had recognized all the sounds and had tracked at least 85% of each sound's duration. There were no false-alarm sound hypotheses. However, there was one false-alarm discrepancy, which, for purposes of clarity, was omitted from the trace. In block 3 the testbed's fault-detection claimed that another discrepancy between the STFT and energy-estimator outputs had occurred. The focusing heuristics did select the associated SOU for handling, but in the course of reprocessing in the same manner as in block 2, no new peaks were found. Thus, the discrepancy was disproven.

This detailed trace shows how the architecture components can implement a dual search to find (1) SPAs appropriate to a scenario with real-world sounds and (2) interpretations appropriate to the SPAs' correlates. The components' activation rates in the trace should not, however, be taken as a measure of their individual utilities in the problem of complex signal interpretation. To determine these utilities, our current work is focused on developing two statistical models. One relates acoustic scenario complexity to distortion rates, and the other relates distortion rates to architecture component activation rates. It is our hope that these models not only

[8]In the current implementation, signal data from the current block and the 2 most recent blocks are buffered. Future configurations will have this buffering governed by a parameter.

will determine each IPUS component's utility for various classes of scenarios but also will generate recognition-rate benchmarks for perceptual systems that do not use various IPUS components.

B.8 IPUS and SPA Design

Traditionally the focus in SPA design has been to develop SPAs that extract, as precisely as possible, *all* details of the desired information from the input signals. The motivation for this design paradigm has been that such SPAs could provide precise information that would efficiently constrain interpretation search and produce interpretations with low uncertainty. This strategy is appropriate provided it can be guaranteed that the signal understanding system will not encounter signals which violate the underlying assumptions made in the design of those SPAs. This premise, however, does not appear appropriate for perceptual systems operating in complex environments [16]. Since in such domains the SPA assumptions will often be violated, it seems unreasonable to devote computational resources to the extraction of detailed and precise information that is likely to be misleading.

The IPUS architecture has important implications for SPA design because it encourages the development and application of fast, highly specialized, theoretically sound SPAs for reprocessing in appropriate contexts. IPUS provides a framework for using such SPAs in strategies where the initial signal processing sacrifices detail and precision, which are then sought during the signal re-processing phase when a better assessment of the signal environment is available. The advantage of sacrificing precision and detail in the initial signal processing is two-fold; the initial signal processing can be more computationally efficient and the discrepancy detection following it is not encumbered by needless quantities of detail.

In the course of our own research on the acoustic interpretation testbed, we have developed a novel algorithm [33] for computing an approximation to the STFT. This approximation retains the major features in the regular STFT output but its computation requires essentially no multiplications (a major part of regular STFT computation) and significantly fewer additions than the regular STFT.

B.9 Future Research

In addition to our work on designing new SPAs and on developing statistical relationships among scenarios, distortion rates, and IPUS components' effectiveness, we are extending our testbed's control plans to explore the issue of scaling. Specifically, we are investigating the use of approximate processing and model-learning.

In configuration *C.2*, which is currently under development, the testbed control plans have been changed to accomodate a larger library of 35 real-world sounds with more complicated structure. The strategies in the new control plans still rely on the basic IPUS framework but now incorporate more goal-directed processing of microstreams and do not propagate the contour interpretations in a bottom-up manner to the microstream level. The processing strategies incorporate approximate-knowledge peak clustering algorithms to constrain source-model selection.

The frequency features of the sound models used in the testbed trace were hand-crafted in a time-consuming process. When dealing with environments with large numbers of signal objects, it will be desirable to automate the model-acquisition process. The construction of these models will require the identification of features that avoid distortions caused by SPAs and/or model interactions as much as possible. Research is being done on incorporating the IPUS reprocessing loop into a framework for learning acoustic source models [3].

On initial consideration, it might seem that the time required by multiple reprocessings under IPUS would be unacceptably high in noisy environments. However, because traditional systems *continuously* sample several front-ends' data while IPUS-based systems *selectively* sample several front-end processings' data, the IPUS paradigm should decrease the expected processing time for contexts requiring several independent processing views. We are working on verifying this claim.

B.10 Summary

In this paper we have considered the problem of signal understanding in complex environments involving interacting objects which mask and/or distort data correlates of their respective features. This implies that during its operation, the perceptual system must continually update, in a context-dependent fashion, what feature-set to focus upon and what SPAs to use in order to extract the features' data correlates. It is important to observe that the selection of a particular SPA is determined not only by the subset of features whose data correlates are sought, but also the presence of data unrelated to those features. We have argued that adaptive selection of features and their corresponding SPAs requires sophisticated but principled control of the interactions between the actions of high-level knowledge sources and the actions of SPAs in a signal understanding system. Motivated by this insight, we have formulated the IPUS architecture for the integrated processing and understanding of signals.

IPUS provides a framework for structuring bidirectional interaction between the search for SPAs appropriate to the environment and the search for interpretation models to explain the SPAs' output data. The availability of a formal signal processing theory is an important criterion for determining the architecture's applicability to any particular domain. IPUS allows system developers to organize diverse signal processing knowledge along the lines of formal concepts such as SPA processing models, discrepancy tests, distortion operators, and SPA application strategies. A major contribution of the architecture is to formalize and unify front-end SPA reconfiguration performed for interpretation processes (e.g. differential diagnosis) with that performed for data correlate refinement. under discrepancy diagnosis. This results in a single reprocessing concept driven by the presence of SOUs.

Our sound understanding testbed experiments indicate that the basic functionality of the architecture's components and their interrelationships are realizable. We believe the IPUS architecture is applicable to any signal understanding domains for which the SPAs have a rich underlying theory. This view is supported by the similarities shared between the testbed's acoustic domain theory and that of many other signal domains such as sonar [39], weather radar [9], music [26], and biomedical signals [14].

In conclusion, we have shown how knowledge from formal signal processing theory regarding the effectiveness of specific SPA configurations for particular environments can be used to develop a highly adaptive signal understanding architecture. This architecture tightly integrates the search for the appropriate SPA configuration with the search for plausible interpretations of the SPA output data. In our opinion, this dual search, informed by formal signal processing theory, is a necessary component of perceptual systems that must interact with complex environments.

B.11 Acknowledgments

We would like to acknowledge Norman Carver and Izaskun Gallastegi for their roles in designing and implementing the IPUS architecture's control framework and evidential reasoning capabilities; Malini Bhandaru and Zarko Cvetanović for their contributions to the testbed's early implementation stages; Erkan Dorken for his significant contribution to designing the testbed's SPAs; and Ramamurthy Mani for his help in creating the acoustic database for testbed experiments.

This work was supported by the Rome Air Development Center of the US Air Force Systems Command under contract F30602-91-C-0038, and by the Office of Naval Research under contract N00014-92-J-1450. The content does not necessarily reflect the position or the policy of the Government, and no official endorsement should be inferred.

Bibliography

[1] Bell, B. and Pau, L. F., "Context Knowledge and Search Control Issues in Object-Oriented Prolog-Based Image Understanding," *The Proceedings of the 1990 European Conference on Artificial Intelligence*, 1990.

[2] Bell, B. and Pau, L. F., "Contour Tracking and Corner Detection in a Logic Programming Environment," *IEEE Transactions on Pattern Recognition and Machine Intelligence*, pp. 913–917, August 1990.

[3] Bhandaru, M. K., Draper, B. A., and Lesser, V. R., "Learning Image to Symbol Conversion," pp. 6-9, AAAI Technical Report FS-93-04, 1993.

[4] Bitar, N., Dorken, E., Paneras, D., and Nawab, S. H., "Integration of STFT and Wigner Analysis in a Knowledge-Based Sound Understanding System," *The Proceedings of the 1992 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. IV, , pp. 585–588, San Francisco, March 1992.

[5] Bobick, A. F. and Bolles, R. C., "The Representation Space Paradigm of Concurrent Evolving Object Descriptions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 146–156, February 1992.

[6] Bonissone, P. and Halverson, P., "Time-Constrained Reasoning under Uncertainty," *The Journal of Real-Time Systems*, vol. 2, pp. 25–45, 1990.

[7] Bregman, A., "Auditory Scene Analysis: The Perceptual Organization of Sound," MIT Press, 1990.

[8] Brooks, R., "Intelligence without Representation," *Artificial Intelligence*, vol. 47, no. 1-3, pp. 139–159, January 1991.

[9] Campbell, S. D. and Olson, S. H., "WX1 – An Expert System for Weather Radar Interpretation," in *Coupling Symbolic and Numerical Computing in Expert Systems*, J. S. Kowalik, ed., Elsevier Science Publishers, B. V. (North-Holland), 1986.

[10] Carver, N. and Lesser, V., "The Evolution of Blackboard Control," *Expert Systems with Applications*, Special Issue on The Blackboard Paradigm and Its

Applications, vol. 7, no. 1, pp. 1–30, 1994, (also available as Technical Report 92-71, Computer Science Department, University of Massachusetts, 1992).

[11] Carver, N. and Lesser, V., "A Planner for the Control of Problem-Solving Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, Special Issue on Planning, Scheduling and Control, vol. 23, no. 6, pp. 1519–1536, November/December 1993.

[12] Carver, N. and Lesser, V., "A New Framework for Sensor Interpretation: Planning to Resolve Sources of Uncertainty," *The Proceedings of the 1991 National Conference on Artificial Intelligence (AAAI-91)*, pp. 724–731, Anaheim, California, July 1991.

[13] Claasen, T. and Meclenbrauker, W., "The Wigner Distribution: A Tool for Time-Frequency Signal Analysis," *Phillips J. Res.*, vol. 35, pp. 276–350, 1980.

[14] Dawant, B. and Jansen, B., "Coupling Numerical and Symbolic Methods for Signal Interpretation," *IEEE Transactions on Systems, Man and Cybernetics*, pp. 115–124, Jan/Feb 1991.

[15] De Mori, R., Lam, L., and Gilloux, M.,"Learning and Plan Refinement in a Knowledge-Based System for Automatic Speech Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 289–305, February 1987.

[16] Dorken, E., "Approximate Processing and Knowledge-Based Reprocessing of Non-Stationary Signals," PhD Thesis, Electrical, Computer, and Systems Engineering Dept, Boston University, 1993.

[17] Dorken, E., Nawab, S. H., and Lesser, V., "Extended Model Variety Analysis for Integrated Processing and Understanding of Signals," *The Proceedings of the 1992 IEEE International Conference on Acoustics, Speech and Signal Processing*, vol. V, pp. 73-76, San Francisco, California, March 1992.

[18] Dove, W., *Knowledge-Based Pitch Detection*, PhD Thesis, Electrical Engineering and Computer Science Dept., MIT, 1986.

[19] Draper, B. A., Hanson, A. R., and Riseman, E. M., "Learning Blackboard-Based Scheduling Algorithms for Computer Vision," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 7, no. 2, pp. 309–328, April, 1993.

[20] Erman, L., Hayes-Roth, F., Lesser, V., Reddy, D., "The Hearsay II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *Computing Surveys 12*, vol. 2, pp. 213–253, June 1980.

[21] Giannesini, F., Kanoui, H., Pasero, R., and van Caneghem, M., *Prolog*, Reading, MA: Addison Wesley, 1986.

[22] Grimson, W. E. L. and Lozano-Pérez, T., "Model-Based Recognition and Localization from Sparse Range or Tactile Data," *The International Journal of Robotics Research*, vol. 3, no. 3, pp. 3–36, 1984.

[23] Hayes-Roth, B., Washington, R., Hewett, R., Hewett, M., and Seiver, A., "Intelligent Monitoring and Control," *The Proceedings of the 1989 International Joint Conference on Artificial Intelligence*, pp. 243–249, Detroit, Michigan, August 1989.

[24] Hudlická, E. and Lesser, V., "Modeling and Diagnosing Problem-Solving System Behavior," *IEEE Transactions on Systems, Man and Cybernetics*, Special Issue on Diagnostic Reasoning, vol. 17, no. 3, pp. 407–419, May/June 1987.

[25] Hudlická, E. and Lesser, V., "Meta-Level Control Through Fault Detection and Diagnosis," *The Proceedings of the 1984 National Conference on Artificial Intelligence (AAAI-84)*, pp. 153–161, Austin, Texas, July 1984.

[26] Katayose, H., Kato, H., Imai, M., and Inokuchi, S., "An Approach to an Artificial Music Expert," *Proceedings of the 1990 International Computer Music Conference*, pp. 139–147, 1990.

[27] Klassner, F., Lesser, V., and Nawab, S. H., "Fusing Multiple Reprocessings of Signal Data," *The Proceedings of the SPIE Sensor Fusion VI Conference*, vol. 2059, Boston, Massachusetts, September 1993.

[28] Klassner, F., *Data Reprocessing and Assumption Representation in Signal Understanding Systems*, Technical Report 92-52, Computer Science Department, University of Massachusetts, 1992.

[29] Kohl, C., Hanson, A., and Reisman, E., "A Goal-Directed Intermediate Level Executive for Image Interpretation," *The Proceedings of the 1987 International Joint Conference on Artificial Intelligence*, pp. 811–814, Milan, Italy, August 1987.

[30] Lesser, V., Nawab, S. H., Gallastegi, I., and Klassner, F., "IPUS: An Architecture for Integrated Signal Processing and Signal Interpretation in Complex Environments," *The Proceedings of the 1993 National Conference on Artificial Intelligence (AAAI-93)*, pp. 249–255, Washington, DC, July 1993.

[31] Lesser, V., Nawab, S. H., Bhandaru, M., Cvetanović, Z., Dorken, E., Gallastegi, I., and Klassner, F., "Integrated Signal Processing and Signal Understanding," Technical Report 91-34, Computer Science Dept., University of Massachusetts, 1991.

[32] Maksym, J. N., Bonner, A. J., Dent, C. A., and Hemphill, G. L., "Machine Analysis of Acoustical Signals," *Issues in Acoustic Signal/Image Processing and Recognition*, C. H. Chen, ed, NATO ASI Series, vol. F1, Springer-Verlag, pp. 95–112, 1983.

[33] Nawab, S. H. and Dorken, E., "Efficient STFT Computation Using a Quantization and Differencing Method," *The Proceedings of the 1993 IEEE Conference on Acoustics, Speech and Signal Processing*, vol. 3, pp. 587–590, Minneapolis, Minnesota, April 1993.

[34] Nawab, S. H. and Lesser, V., "Integrated Processing and Understanding of Signals," chapter 6, *Knowledge-Based Signal Processing*, pp. 251–285, A. Oppenheim and H. Nawab, eds., Prentice Hall, New Jersey, 1991.

[35] Nawab, S. H. and Lesser, V., "High-Level Adaptive Signal Processing", *Northeast AI Consortium Final Report*, Technical Report RADC-TR-90-404, vol. 17, Rome Laboratories, Griffiss Air Force Base, NY, 13441-5700, Dec 1990.

[36] Nawab, S. H. and Quatieri, T., "Short-Time Fourier Transform," *Advanced Topics in Signal Processing*, Prentice Hall, New Jersey, 1988.

[37] Nawab, S. H., Lesser, V., and Milios, E., "Diagnosis Using the Underlying Theory of a Signal Processing System," *IEEE Transactions on Systems, Man and Cybernetics*, Special Issue on Diagnostic Reasoning, vol. 17, no. 3, pp. 369-379, May/June 1987.

[38] Newell, A. and Simon, H., "GPS: A Program that Simulates Human Thought." *Computers and Thought*, Feigenbaum and Feldman, eds., McGraw-Hill, pp. 279–293, 1969.

[39] Nii, H., Feigenbaum, E., Anton, J., and Rockmore, A., "Signal-to-Symbol Transformation: HASP/SIAP Case Study," *AI Magazine*, vol. 3, Spring 1982.

[40] Oppenheim, A. V. and Schafer, R. W., *Discrete-Time Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.

[41] Peng, Y. and Reggia, J. "Plausibility of Diagnositic Hypotheses: The Nature of Simplicity," *The Proceedings of the 1986 National Conference on Artificial Intelligence (AAAI-86)*, pp. 140–145, Philadelphia, Pennsylvania, July 1986.

[42] Seborg, D., et al., "Adaptive Control Strategies for Process Control: A Survey," *AIChE Journal*, vol. 32, no. 6, pp. 881–913, June 1986.

[43] Swain, M. and Stricker, M., eds. *Promising Directions in Active Vision,* NSF Active Vision Workshop, Technical Report CS 91-27, Computer Science Department, University of Chicago, 1991.

[44] Williams, M., "Hierarchical Multi-Expert Signal Understanding," *Blackboard Systems*, R. Engelmore and A. Morgan, eds., Addison-Wesley, 1988.

Appendix C

C++ IPUS Users' Manual

C++ IPUS Platform User's Manual

Joseph M. Winograd

Knowledge-Based Signal Processing Laboratory
Dept. of Electrical, Computer, and Systems Engineering
Boston University
winograd@bu.edu

C.1 Overview

The C++ IPUS platform is an environment for the development of signal understanding applications using the approaches described in [1, 2, 3]. This document assumes familiarity with those works.

A scripting language layer on top of C++ is provided for the design of testbed applications and hides many of the semantics particular to C++ . Knowledge of the C language [4], an understanding of object-oriented design, and access to a general source of C++ information [5, 6] should suffice the application designer.

The platform is comprised of a library of base objects which together provide the full set of IPUS architectural features. The objects are implemented as C++ classes and take advantage of the language features of inheritance, polymorphism, encapsulation, and information hiding. Except where noted, the base classes described here are *abstract* and cannot be directly instantiated. They embody high-level abstractions of the objects that they represent and have *pure virtual* methods. Pure virtual methods are methods that are known to be required for the object to be instantiated, but for which no generalized definition can be given. To create an instance of an abstract base class, a new *concrete* class must be derived from the base class, for which the pure virtual (or *required*) slots and methods are defined.

The construction of an IPUS system using this platform is a process of deriving concrete classes with specialized behaviors from the base classes. This document presents those base classes and demonstrates the process by which the elements of an IPUS application are derived and integrated into a complete signal understanding system.

C.2 IPUScript Grammar

IPUScript is a scripting language based on the macro facility of the C++ preprocessor. It simplifies the task of writing applications with the IPUS platform by providing a small set of programming constructs which hide much of the mechanics of C++ programming. It thereby allows the application developer to focus on aspects of the design process more directly related to the task of signal understanding.

This section presents a production grammar describing the IPUScript language. The grammar produces a superset of valid IPUScript applications, and is included more as a general reference for the IPUS application developer than for parser generation. It may be considered as a set of extensions to the C++ grammar in [5], since any legal C++ construct may be found within an IPUScript application. Production rules mentioned herein but not defined refer to existing C++ rules.[1]

At the highest level, an IPUScript application has the following structure

ipuscript-application:
 declaration-segment
 definition-segment

The *declaration-segment* includes the interface specification for the application and is typically placed in a C++ header file. It has the structure

declaration-segment:
 `#include "ipuscript.h"`
 `BEGIN_DECLARE_SEGMENT`
 declaration-list$_{\text{opt}}$
 `END_DECLARE_SEGMENT`

declaration-list:
 short-declaration | *long-declaration*
 declaration-list$_{\text{opt}}$

A declaration derives a concrete class from one of the IPUS base classes. Declarations have the form

short-declaration:
 `DECLARE_`*base-class-name*`(` *class-name* `)`

or

[1]The reader is urged to use extra care when examining this document in conjunction with [4, 5] since a different convention for describing production rules is used in those manuscripts.

long-declaration:
    BEGIN_DECLARE_*base-class-name*(*class-name*)
    declaration-body
    END_DECLARE_*base-class-name*

declaration-body:
    data-member-declaration-list$_{\text{opt}}$
    member-function-declaration-list$_{\text{opt}}$
    optional-method-declaration-list$_{\text{opt}}$

The *short-declaration* form may be used when no data members, member functions, or optional methods are included in the declaration. *base-class-name* is the name of the base class from which the declared class is derived, and has the structure

base-class:
    IPUS | BLACKBOARD | HYPOTHESIS | SOU | NP_PLAN | SUBGOAL |
        P_PLAN | REFOCUS_UNIT

class-name is the name given to the declared class and must be a valid C++ identifier. *data-member-declaration-list* and *member-function-declaration-list* are lists of declarations for public data members and member functions that are specific to the derived class and follow standard C++ syntax. *optional-method-declaration-list* has the structure

optional-method-declaration-list:
    optional-method-declaration
    optional-method-declaration-list$_{\text{opt}}$

optional-method-declaration:
    DECLARE_*optional-method*

optional-method:
    VERIFY | PRINT | PLOT | IN_CONSTRAINTS | PRECONDITION | IN_BINDINGS |
        VARIABLE_FOCUS | SUBGOAL_FOCUS | MATCH_FOCUS | FAILURE |
        SATISFACTION | OUT_CONSTRAINTS

For each declared optional method, an associated definition must appear in the definition segment. Only a subset of all optional methods listed in the grammar specification above are applicable for any particular base class. These associations are explained in Secs. C.3-C.10. Required methods for abstract base classes should not be explicitly declared, nor should optional methods for which the default behavior of the base class is desired.

The *definition-segment* specifies values for inherited slots and class specific method behaviors for derived classes. It is typically placed in a separate source file from the *definition-segment*, which must be included (with `#include`) at its beginning. The *definition-segment* has the structure

definition-segment:
 BEGIN_DEFINE_SEGMENT
 definition-list$_{\text{opt}}$
 END_DEFINE_SEGMENT

definition-list:
 definition
 definition-list$_{\text{opt}}$

definition:
 DEFINE_*base-class*(*class-name, required-slot-values*)
 method-definition$_{\text{opt}}$

required-slot-values:
 slot-value | *slot-value, required-slot-values*

method-definition-list:
 method-definition
 method-definition-list$_{\text{opt}}$

method-definition:
 BEGIN_DEFINE_*base-class_method-name*(*class-name*)
 statements$_{\text{opt}}$
 END_DEFINE_*base-class_method-name*

method-name:
 VERIFY | SUMMARIZE | PRINT | PLOT | GRAMMAR | IN_CONSTRAINTS |
  PRECONDITION | IN_BINDINGS | VARIABLE_FOCUS | SUBGOAL_FOCUS |
  MATCH_FOCUS | FAILURE | SATISFACTION | OUT_CONSTRAINTS |
  ACTION | CONDITION | HANDLER | REMOVAL

The *definition-segment* must contain a *definition* for each derived class declared in the *declaration segment.* Each *definition* must contain a *slot-value* for each required slot of the base class. Each *slot-value* is an expression which evaluates to a constant value of the slot's data type. Additionally, a *definition* must contain a *method-definition* for each required method and for each optional method which was explicitly declared in the *declaration* of the class. Slot and method specification is described in greater detail in Secs. C.3-C.10.

C.3 IPUS Applications

Sec. C.2 presented a set of grammatical rules which describe the scope of valid IPUScript language constructs. In this section we discuss the development of IPUS applications from a design oriented point of view.

A complete IPUS system incorporates a RESUN control planner [3], a blackboard, a problem solving model (PSM), and libraries of hypotheses, SOUs, plans, subgoals, and refocus units. In the IPUS platform, the nucleus of this system is the *IPUS application* class. It contains the control planner and blackboard and serves as the environment in which all elements of the IPUS system interact. It also provides the interface by which software external to the IPUS system directs and receives the results of the signal analysis. The following sections describe the derivation of a concrete IPUS application class and its use within a larger software system.

C.3.1 Deriving IPUS Applications

The process of deriving a concrete IPUS application class is primarily one of unifying the disparate elements of the IPUS system into a single unit. In addition, a *region* class must be derived which defines the structure of the signal data space.

A derived IPUS application generally introduces no data members or member functions, and no optional methods are supported in IPUScript for the `IPUS` base class. As detailed in Table C.1, its required slots specify the name of the blackboard class to be instantiated by the system and the total number of derived plan and subgoal classes available to the planner. The required methods inform the planner of the class ID of each plan, subgoal, and refocus unit used in the system by "registering" them with the IPUS application. An example of the derivation of an IPUS application is given below.

```
// This code fragment is from the C++ header file cat.h
#include "ipuscript.h"
        ⋮
// CAT_TAG, PLAN_TAG, SUBGOAL_TAG, and REFOCUS_UNIT_TAG are defined in ipuscript.h.
#define CAT_ID (CAT_TAG + 1)

#define PID_SOLVE_PROBLEM (PLAN_TAG + 1)
#define PID_INITIALIZE_PSM (PLAN_TAG + 2)
        ⋮
#define PID_GET_SOLUTION (PLAN_TAG + 25)
#define TOTAL_PLAN_COUNT 25

#define SGID_HAVE_PSM_INITIALIZED (SUBGOAL_TAG + 1)
        ⋮
#define SGID_HAVE_SOLUTION (SUBGOAL_TAG + 15)
#define TOTAL_SUBGOAL_COUNT 15

// Only one refocus unit is defined.
```

Base Class Name	*REQUIRED SLOTS*		
	name	*type*	*description*
IPUS	ID	int	Class ID.
	PLAN_COUNT	int	Total number of defined plans (non-primitive and primitive).
	SUBGOAL_COUNT	int	Total number of defined subgoals.
	BLACKBOARD_CLASS	ident	The name of the concrete blackboard class to be used with this IPUS application.

Base Class Name	*REQUIRED METHODS*	
	name	*description*
IPUS	REGISTER_PLANS	Register class IDs of all plans.
	REGISTER_SUBGOALS	Register class IDs of all subgoals.
	REGISTER_REFOCUS_UNITS	Register class IDs of all refocus units.

Base Class Name	*OPTIONAL METHODS*		
	name	*description*	*default behavior*
IPUS	-	-	-

Table C.1: IPUScript slot and method specifications for the IPUS base class.

```
#define RUID_TIMER_UNIT (REFOCUS_UNIT_TAG + 1)

BEGIN_DECLARATION_SEGMENT
// Declaration of Cat - a concrete IPUS class.
DECLARE_IPUS( Cat )
    ⋮
END_DECLARATION_SEGMENT
// End of code fragment from file cat.h

// This code fragment is from the C++ source file cat.C
BEGIN_DEFINITION_SEGMENT
// Definition of Cat.
DEFINE_IPUS( Cat, CAT_ID, TOTAL_PLAN_COUNT, TOTAL_SUBGOAL_COUNT, CatBlackboa

BEGIN_DEFINE_IPUS_REGISTER_PLANS( Cat )
    REGISTER( SolveProblem, PID_SOLVE_PROBLEM )
    REGISTER( InitializePSM, PID_INITIALIZE_PSM )
        ⋮
    REGISTER( GetSolution, PID_GET_SOLUTION )
```

```
END_DEFINE_IPUS_REGISTER_PLANS

BEGIN_DEFINE_IPUS_REGISTER_SUBGOALS( Cat )
    REGISTER( HavePSMInitialized, SGID_HAVE_PSM_INITIALIZED )
        ⋮
    REGISTER( HaveSolution, SGID_HAVE_SOLUTION )
END_DEFINE_IPUS_REGISTER_SUBGOALS

BEGIN_DEFINE_IPUS_REGISTER_REFOCUS_UNITS( Cat )
    REGISTER( TimerUnit, RUID_TIMER_UNIT )
END_DEFINE_IPUS_REGISTER_REFOCUS_UNITS
    ⋮
END_DEFINITION_SEGMENT
// End of code fragment from file cat.C
```

Deriving the Region Class

A region class must be defined as a companion to the IPUS application. A region class is a passive structure whose data members specify a specific region of the signal data space. For example, in a sound understanding application the region structure could contain three slots indicating the starting and ending times of a sound segment and a tag indicating the source of the data. In a radar signal analysis application the region structure might contain the time stamp of the return, the sensor with which it was collected, and the range and azimuth boundaries of the signal data.

Only a single region class may be derived for an IPUS application. It is currently required that the region class be given the class name `Region`. The region class has no required slots or optional methods, and has a single required method—`PRINT`. The `PRINT` method must be defined, but is called only when the `TRACE_PSM` option of the trace output facility is used. `PRINT` receives a single parameter `stream` of type `ostream` which is an open output stream to which the trace output should be written. An example region class for a radar clutter analysis application is shown below.

```
// Code fragment from declaration segment of file cat.h
    ⋮
BEGIN_DECLARE_REGION( Region )
    int timeSlice,
        rangeSpace[2],
        azimuthSpace[2];
    String fileName;
END_DECLARE_REGION
```

```
// End code fragment from file cat.h
// Code fragment from definition segment of file cat.C
     :
BEGIN_DEFINE_PRINT_REGION
    stream << timeSlice << ", [("
        << rangeSpace[0] << "," << azimuthSpace[0] << ")-("
        << rangeSpace[1] << "," << azimuthSpace[1] << ")]";
END_DEFINE_PRINT_REGION
// End code fragment from file cat.C
```

C.3.2 Using IPUS Applications

IPUS applications created with the testbed are not themselves executable programs, but are signal understanding objects which may become modular components of a larger overall software system. The IPUS application, therefore, provides a public interface by which signal understanding requests may be made and the results provided to other system components. The methods listed below are those provided by the IPUS base class and are inherited by all derived IPUS applications.

Public Methods

`IPUS( TraceType tt = TRACE_NOTHING, ostream& stream = cout ):`
The IPUS class constructor takes two parameters which control the trace output facility of the testbed. `tt` may take on any combination of the labels `TRACE_NOTHING`, `TRACE_PLANS`, `TRACE_BLACKBOARD`, `TRACE_SUBGOALS`, `TRACE_PSM`, `TRACE_SOU`, `TRACE_ALL`. These labels may be combined using logical OR to produce a fully customized trace output. Trace output may be directed to any open output stream by passing that stream as the `stream` parameter.

`int fail();`
Return a boolean value indicating whether the most recent call to `understand` halted due to an internal error condition. Run-time errors generate an immediate error message on the **cerr** stream, an internal logging of the error message (see `why` below), and the halting of the control planner when the next focus point completes.

`BlackboardPtr getBlackboardPtr();`
Return a pointer to the blackboard instance being used by the IPUS system.

`uint isA();`
Return the class ID.

`String nameOf();`

Return the class name.

`HypothesisPtr understand( void *input ):`

Perform signal understanding. The top level plan (i.e. the plan with class ID `PLAN_TAG+1` is instantiated and run with `input` as its input. The main control loop of the RESUN planner is outlined in Fig. C.3.1. When that

```
put SolveProblem plan instance on currentFocusPoints queue
repeat
  repeat
    dequeue focusPoint from currentFocusPoints queue
    if focusPoint is non-primitive do
      if focusPoint is not a plan version
        focus on focusPoint input variables
        set focusPoint to first plan version
        place alternate plan versions at head of currentFocusPoints queue
      end if
      execute next step of focusPoint grammar
      focus on new subgoals
      for each new subgoal do
        match new plan instances to new subgoal
        focus on matching plan instances
        place resulting plan instances on nextFocusPoints queue
      end for
      if focusPoint is completed then
        update planning structure
      end if
      invoke or remove all applicable refocus units
    else { focusPoint is primitive }
      focus on focusPoint input variables
      set focusPoint to first plan version
      place alternate plan versions at head of currentFocusPoints queue
      perform focusPoint primitive action
      invoke blackboard summarization
      check satisfaction conditions of all current subgoals
      check failure conditions of all plan versions
      update planning structure
      invoke or remove all applicable refocus units
    end if
  until currentFocusPoints is empty
  swap currentFocusPoints and nextFocusPoints
until currentFocusPoints is empty
```

Figure C.3.1: The RESUN planner control loop.

plan completes or the planner halts, the output from that plan is returned.

`String why();`

Return a string containing one or more reasons why the planner halted without completing the signal understanding task.

C.3.3 Problem Solving Model

The problem solving model, as instantiated in the generic IPUS architecture [1], has been incorporated into the IPUS testbed in the form of the concrete `PSM` class. The PSM maintains references to important hypotheses on the blackboard and simplifies the control plans' task of selecting SOUs to solve. Its structure and relation to the blackboard is shown in Fig. C.3.2.

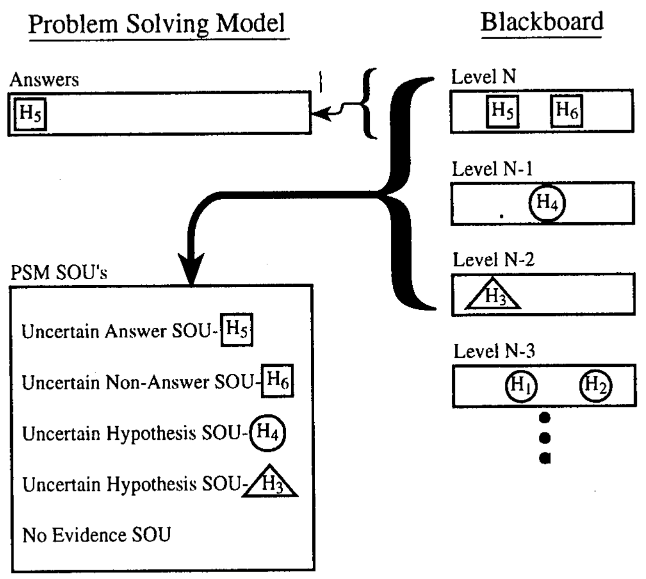


Figure C.3.2: The problem solving model (PSM).

The blackboard summarization performed by the PSM incorporates a system for calculating numerical hypothesis ratings based on [1]. The system produces a summary for each hypothesis by recursively summarizing each of its supporting hypotheses, and combining the uncertainty of the support with the hypothesis' own uncertainty.

The PSM is comprised of two related elements—the *answer list* and the *PSM SOU list.* The answer list is a list of pointers to those hypotheses on the blackboard that represent potential high-level solutions to the signal understanding task currently underway. A hypothesis is considered a potential answer if it is at a high level of the blackboard and has a high rating. The PSM SOU list is a sorted list of pointers to PSM SOU instances. PSM SOUs are symbolic representations of uncertainty in the overall signal understanding task; as its state is reflected on the blackboard.

There are currently four types of PSM SOUs defined in the testbed. *Uncertain answer SOUs* are created for all hypotheses on the answer list of the PSM. *Uncertain non-answer SOUs* are created for those hypotheses that are high enough on the blackboard to be considered answers but are not highly rated. *Uncertain hypothesis SOUs* are created for hypotheses in the mid-to-upper levels of the blackboard that cannot be considered potential answers. Each of these PSM SOUs contain a pointer to the hypothesis to which they refer. *No evidence SOUs* represent the overall uncertainty related to signal data that has yet to be collected by the system and placed on the first level of the blackboard. Instead of referring to a hypothesis instance, it contains a region class instance describing the signal data region not yet considered. PSM SOUs are ordered on the PSM SOU list according to the priority system described below in the Public Slots section.

The PSM is a concrete class and need only be instantiated and "installed" into the application object from within a plan schema or primitive action. This operation is usually the first subgoal of the top level plan [1]. At instantiation, the behavioral properties of the PSM are set through the constructor's parameter list described below. The PSM is initialized by instantiating a No evidence SOU over the first signal region of interest and placing it on the PSM SOU list. The PSM is installed with the `installPSM` utility function described in Sec. C.12. Once installed, the PSM is updated after each primitive action completes.

Public Slots

`HypothesisPtrList answerList;`

A list of pointers to hypothesis on blackboard levels `bottomAnswerLevel` through `topAnswerLevel` that have a rating not less than `minimumAnswerRating`.

`PSMSOUPtrList psmSOUList;`

A sorted list of pointers to PSM SOU instances. PSM SOUs are maintained for all hypotheses on blackboard levels `bottomHypothesisLevel` through `topAnswerLevel`. Hypotheses on levels `bottomHypothesisLevel` through `topHypothesisLevel` are represented by `UncertainHypothesisSOUs`. Hypotheses on levels `bottomAnswerLevel` through `topAnswerLevel` are represented by `UncertainAnswerSOUs` if their ratings are not less than `minAnswerRating`, `UncertainNonAnswerSOUs` if their rating is less than `minAnswerRating` but not less than `minNonAnswerRating`, or `UncertainHypothesisSOUs` otherwise. `NoEvidenceSOUs` are neither created, removed, nor modified by the PSM, and are solely under the control of the plans. The list is sorted by a priority structure in which the SOUs are first ordered by type and then by blackboard level. `UncertainAnswerSOUs` precede `UncertainNonAnswerSOUs` followed (in order) by `UncertainHypothesisSOUs` and `NoEvidenceSOUs`. Within an SOU type, SOUs referring to hypotheses on higher levels of the blackboard precede those referring to lower level hypotheses.

Public Methods

```
PSM( int bottomHypothesisLevel, int topHypothesisLevel, int bottomAnswerLevel,

      int topAnswerLevel, double minAnswerRating = 0.5, double
      minNonAnswerRating
      = 0.5 );
```

PSM constructor. Sets up the behavior of the PSM by assigning values to its control variables. The effects of these variables are described in the text above.

C.4 Blackboards

A blackboard is a hierarchy of a fixed number of blackboard levels. Each blackboard level is a list of pointers to hypothesis instances.

C.4.1 Deriving Blackboards

The derivation of a concrete blackboard class requires the specification of a class ID, the number of blackboard levels, and optionally a method for asserting hypothesis posting restrictions (see Table C.2).

Base Class Name	*REQUIRED SLOTS*		
	name	*type*	*description*
`BLACKBOARD`	`ID`	int	Class ID.
	`LEVELS`	int	Number of blackboard levels.

Base Class Name	*REQUIRED METHODS*	
	name	*description*
`BLACKBOARD`	-	-

Base Class Name	*OPTIONAL METHODS*		
	name	*description*	*default behavior*
`BLACKBOARD`	`VERIFY`	Verify that a posted hypothesis is "valid" wrt some general policy of blackboard usage.	Allow any hypothesis to be posted.

Table C.2: IPUScript slot and method specifications for the `BLACKBOARD` base class.

The `ID` slot holds an integer value added to the predefined `BLACKBOARD_TAG` constant. The `LEVELS` slot specifies the number of separate blackboard levels. The `VERIFY` method, if defined, is called whenever a new hypothesis is posted to the blackboard and may veto its posting if the hypothesis is deemed harmful to the system integrity in some way. This allows the enforcement of hypothesis type restrictions with respect to specific blackboard levels and the application of a generalized test which may ensure that only "well-formed" hypotheses are posted. The `VERIFY` method receives a single argument `hypothesis` of type `HypothesisPtr` and returns a boolean value indicating whether the structure of the hypothesis has been verified. If the hypothesis is found to be invalid, an internal error may be catalogued using the IPUScript utility function postError described in Sec. C.12. An example of the derivation of a concrete blackboard class is shown below.

```
// Code fragment from the file cat.h
    ⋮
#define CAT_BLACKBOARD_ID (BLACKBOARD_TAG + 1)

#define RETURN_LEVEL (1)
#define MAP_LEVEL (2)
#define REGION_LEVEL (3)
#define PATCH_LEVEL (4)
#define TOP_LEVEL PATCH_LEVEL

#define RETURN_HYP (HYPOTHESIS_TAG + 1)
#define MAP_HYP (HYPOTHESIS_TAG + 2)
#define REGION_HYP (HYPOTHESIS_TAG + 3)
#define PATCH_HYP (HYPOTHESIS_TAG + 4)
    ⋮
BEGIN_DECLARATION_SEGMENT
    ⋮
BEGIN_DECLARE_BLACKBOARD( CatBlackboard )
    DECLARE_VERIFY
END_DECLARE_BLACKBOARD
    ⋮
// End of code fragment from cat.h

// Code fragment from the definition segment of file cat.C
    ⋮
DEFINE_BLACKBOARD( CatBlackboard, CAT_BLACKBOARD_ID, TOP_LEVEL )

BEGIN_DEFINE_BLACKBOARD_VERIFY( CatBlackboard )
    switch (hypothesis->level()) {
        case RETURN_LEVEL: {
            return (hypothesis->isA() == RETURN_HYPOTHESIS);
            break;
        }
        case MAP_LEVEL: {
            return (hypothesis->isA() == MAP_HYPOTHESIS);
            break;
        }
        case REGION_LEVEL: {
                return (hypothesis->isA() == REGION_HYPOTHESIS);
                break;
            }
            case PATCH_LEVEL: {
                return (hypothesis->isA() == PATCH_HYPOTHESIS);
                break;
            }
            default:  {
                return 0;
                break;
            }
        }
    END_DEFINE_BLACKBOARD_VERIFY
        ⋮
    // End of code fragment from cat.C
```

C.4.2 Using Blackboards

The blackboard is accessible to all plan, subgoal, and refocus unit methods as well as software components external to the IPUS system through the public `getBlackboardPtr` method of the IPUS application class. Access is simplified within the IPUS system by the use of the IPUScript utility functions described in Sec. C.12. Hypotheses are added to the blackboard with the `addHypothesis` method.

The levels themselves are unsorted lists, and the viewing and changing of hypotheses is performed through the use of `LevelIterator` objects. `LevelIterators` may be created by calls to the public `getLevelIterator` method of the blackboard class. The use of lists and iterators are described in Sec. C.11. `LevelIterators` differ from the generic iterator only in that they have no `remove` method.

The blackboard base class provides the following methods

Public Methods

```
void addHypothesis( const LevelID level,
                    HypothesisPtr hypothesis );
```

Add `hypothesis` to the specified blackboard level (if allowed by the internal `VERIFY` method described below). Blackboard levels are numbered starting with 1.

```
LevelIterator getLevelIterator( const uint level );
```

Return an iterator over the specified blackboard level. `LevelIterators` provide all the functionality of iterators as described in Sec. C.11.2, except no `remove` method is provided.

`uint isA();`
: Return the class ID.

`String nameOf();`
: Return the class name.

`uint levels();`
: Return the number of blackboard levels.

C.5 Hypotheses

Hypothesis objects represent interpretive assertions made by knowledge sources. They are placed on the blackboard where they may be examined and modified by other knowledge sources.

C.5.1 Deriving Hypotheses

Concrete hypothesis classes are derived from the base `HYPOTHESIS` class by specifying a class ID, declaring the hypothesis' data slots, and optionally specifying methods for printing and plotting of the object as part of the IPUS application trace output. The IPUScript specifications for inherited slots and methods are given in Table C.3. Each derived hypothesis class is usually restricted to being placed on a particular

Base Class Name	*REQUIRED SLOTS*		
	name	*type*	*description*
`HYPOTHESIS`	`ID`	int	Class ID.

Base Class Name	*REQUIRED METHODS*	
	name	*description*
`HYPOTHESIS`	-	-

Base Class Name	*OPTIONAL METHODS*		
	name	*description*	*default behavior*
`HYPOTHESIS`	`PRINT`	Print hypothesis to a text stream.	Print class name, instance ID, and rating.
	`PLOT`	Plot hypothesis graphically.	Do nothing.

Table C.3: IPUScript slot and method specifications for the `HYPOTHESIS` base class.

blackboard level, and in many IPUS systems this is a one-to-one association. These restrictions may be enforced with the blackboard class' VERIFY method.

The most important elements of each derived hypothesis class are the object specific data slots. These slots are set and modified by knowledge sources (in primitive actions) to encode their interpretations of lower level hypotheses as well as high-level expectations. Object specific data slots are enumerated within the *data-member-declaration-list* field of the class declaration (see Sec. C.2). They are declared using the standard data member declaration syntax of C++ and should be public members (explicit use of the public keyword is not required within IPUScript).

The PRINT and PLOT methods are used only by the platform trace facility (see Sec. C.3.2) and have no effect on signal understanding per se. When the TRACE_BLACKBOARD portion of the trace facility is enabled, the PRINT method of each hypothesis on the blackboard is called in turn, starting from the first hypothesis on the highest level of the blackboard and continuing through to the last hypothesis on the lowest level. PRINT is called with a single parameter named stream of type ostream, which is an open stream to which the trace output is to be written. The name of the hypothesis class, a unique instance identifier for the object, and the hypothesis' rating is sent to the stream prior to the PRINT method being called. Some experimenting may be necessary with output formatting to produce an attractive (i.e. readable) trace output when it is desired to show the contents of the hypothesis' slots as part of the printed output. The PRINT method may also be used to write data to an external file for debugging purposes.

The PLOT method is provided to support the design of extensions to the platform which produce a graphical display. The PLOT method of each hypothesis on the blackboard is called after each primitive action is executed.

An example of the derivation of a hypothesis class from a radar signal analysis application is shown below.

```
// Code fragment from header file cat.h
     ⋮
#define RETURN_HYPOTHESIS (HYPOTHESIS_TAG + 1)
     ⋮
#define NUMBER_OF_RANGE_GATES (100)
#define NUMBER_OF_AZIMUTH_GATES (100)
     ⋮
BEGIN_DECLARATION_SEGMENT
     ⋮
BEGIN_DECLARE_HYPOTHESIS( ReturnHypothesis )
    DECLARE_PRINT
    Region region;
    double data[ NUMBER_OF_RANGE_GATES ][ NUMBER_OF_AZIMUTH_GATES ];
```

```
END_DECLARE_HYPOTHESIS
    ⋮
// End of code fragment from file cat.h

// Code fragment from definition segment
// in source file cat.C
    ⋮
DEFINE_HYPOTHESIS( ReturnHypothesis, RETURN_HYPOTHESIS )
BEGIN_DEFINE_HYPOTHESIS_PRINT( ReturnHypothesis )
    stream << data[0][0] << ", " << data[1][0]
    << ", " << data[2][0] << "...";
END_DEFINE_HYPOTHESIS_PRINT
    ⋮
// End of code fragment from file cat.C
```

C.5.2 Using Hypotheses

A hypothesis is a passive entity and has only inert public methods. It inherits a number of slots that are used by the knowledge sources to maintain the hypothesis' inferential relationship to other hypotheses on the blackboard, as well as the uncertainty associated with those inferences. Hypothesis slots are never altered by the control planner or the blackboard. The slots described below belong to all derived hypothesis classes and encode information used throughout the IPUS system.

Public Slots

`HypothesisPtrList explanationList;`
: A list of pointers to higher level hypotheses that have been created from inferences supported by this hypothesis. This list must be maintained by the knowledge sources in their `ACTION` methods.

`SOUPtrList souList;`
: A list of pointers to the SOU instances attached to this hypothesis. SOUs are created and attached to hypotheses and also removed from them by knowledge sources. Use of SOUs is described in Sec. C.6.2

`HypothesisPtrList supportList;`
: A list of pointers to lower level hypotheses that support the inference on which this hypothesis is based. This list must be maintained by the knowledge sources in their `ACTION` methods.

`Summary summary;`

A numeric summarization of the uncertainty associated with this hypothesis and its support. The summary is maintained by the PSM (if installed) and updated every time a primitive action is run. The summary ratings are calculated by the algorithm described in Sec. C.3.3.

<u>**Public Methods**</u>

`uint isA();`

Return the class ID.

`uint level();`

Return the number of the blackboard level on which this hypothesis resides. This value comes from a protected slot which is set by the blackboard method `addHypothesis` when the hypothesis is posted.

`String nameOf();`

Return the class name.

C.6 SOUs

An SOU (or *source of uncertainty*) is a symbolic expression of a specific kind of uncertainty associated with a hypothesis. SOU instances are created and attached to hypotheses by knowledge sources, and serve two distinct purposes in the IPUS testbed. SOUs are used in the process of blackboard summarization, where they impact on the ratings summary of the hypothesis to which they are attached and its explanations. SOUs may also be used to guide focusing decisions within the planning process.

C.6.1 Deriving SOUs

SOUs are all derived from the base `SOU` class. The slot and method specifications for a derived SOU class are shown in Table C.4. Each derived SOU class must be assigned a unique class ID number. This ID is derived by adding a small integer to the predefined `SOU_ID` constant. SOUs must also be assigned an integral priority value. SOU priorities are used for ordering SOUs on the sorted `souList` associated with each hypothesis. Smaller numbers indicate higher priority. This ordering can be used to simplify control decisions when multiple SOUs are associated with a single hypothesis.

A concrete SOU class also requires the specification of a `SUMMARIZE` method. This method is called by the PSM during blackboard summarization and assigns a value to the SOU's internal `rating` slot. The `rating` is a double precision floating point number in the range $[0, 1]$ which indicates the the amount of uncertainty attributed

Base Class Name	*REQUIRED SLOTS*		
	name	*type*	*description*
`SOU`	`ID`	int	Class ID.
	`PRIORITY`	int	Priority for sequencing of SOU lists.

Base Class Name	*REQUIRED METHODS*	
	name	*description*
`SOU`	`SUMMARIZE`	Apply SOU uncertainty to hypothesis summary.

Base Class Name	*OPTIONAL METHODS*		
	name	*description*	*default behavior*
`SOU`	`PRINT`	Print SOU to a text stream.	Print SOU name.

Table C.4: IPUScript slot and method specifications for the `SOU` base class.

to the hypothesis by this SOU. Larger ratings indicate greater uncertainty. The method by which individual SOU ratings are combined to form overall hypothesis ratings is described in Sec. C.3.3.

The optional SOU `PRINT` method is used by the platform trace facility for displaying the SOU on a text stream. Its structure is the same as that described for the hypothesis class' print method in Sec. C.5.1. An example of the derivation of a concrete SOU class is given below. A number of derived SOU classes are pre-defined in the platform for use in applications and are described in Table C.5.

```
// Code fragment from header file cat.h
        ⋮
#define PARTIAL_VERIFICATION_SOU (SOU_TAG + 20)
        ⋮
BEGIN_DECLARATION_SEGMENT
        ⋮
BEGIN_DECLARE_SOU( PartialVerificationSOU )
    DECLARE_SUMMARIZE
END_DECLARE_SOU
        ⋮
// End of code fragment from file cat.h

// Code fragment from definition segment in source file cat.C
```

```
        ⋮
DEFINE_SOU( PartialVerificationSOU, PARTIAL_VERIFICATION_SOU, 110 )
BEGIN_DEFINE_SOU_SUMMARIZE( PartialVerificationSOU )
    rating = 0.5;
END_DEFINE_SOU_SUMMARIZE
        ⋮
// End of code fragment from file cat.C
```

C.6.2 Using SOUs

SOUs are added and removed from hypotheses by knowledge sources within their primitive actions. SOU instances are created with direct calls to the C++ **new** operator. The SOU constructor requires a single parameter that is a pointer to the hypothesis to which the SOU will be attached. The SOU must be subsequently attached to the hypothesis by adding a pointer to it onto the `souList` data member of the hypothesis. This action requires that a pointer to the hypothesis itself be available.

An SOU may be removed from a hypothesis by removing its pointer from the hypothesis' `souList` data member. It must also be **deleted** as required by C++ for correct free-store usage. Addition and removal from the `souList` is performed using the public `List` class methods as described in Sec. C.11.

C.7 Non-Primitive Plans

Plan objects serve a single purpose in the IPUS testbed—they provide a means of meeting a subgoal of the problem solving task. Non-primitive plans meet a subgoal by decomposing it into a sequence of simpler subgoals. The sequencing of these subgoals is encoded as a series of steps in an algorithmic structure called a *plan schema*. In addition to the plan schema, a variety of additional methods may be defined for a non-primitive plan to assist the control planner in selecting the best plan to meet a particular subgoal and to integrate the plan instance with the other plans being carried out in parallel within the testbed. In the following sections we discuss the mechanics of deriving non-primitive plan classes and demonstrate several of the features provided for specializing their behavior.

C.7.1 Deriving Non-Primitive Plans

Table C.6 lists the derived slots and methods available for non-primitive plans. The `ID` slot assigns a unique plan class identifier that is used throughout the platform

Derived Class Name	*Class ID*	*Priority*	*Rating*	*Purpose*
`NoExplanationSOU`	`NO_EXPLANATION_SOU`	100	0.5	No explanations have been identified.
`NoSupportSOU`	`NO_SUPPORT_SOU`	110	0.5	No support has been identified.
`Partial SupportSOU`	`PARTIAL_ SUPPORT_SOU`	120	0.5	Existence of required evidence remains unverified.
`Missing EvidenceSOU`	`MISSING_ EVIDENCE_SOU`	200	0.5	Expected evidence has been found to be missing.
`Support ExclusionSOU`	`SUPPORT_ EXCLUSION_SOU`	200	0.5	Evidence to support an expectation has been found to be missing.
`Support LimitationSOU`	`SUPPORT_ LIMITATION_SOU`	200	0.5	Support is limited until the results of further processing are obtained.
`Alternative ExtensionSOU`	`ALTERNATIVE_ EXTENSION_SOU`	200	0.5	Competing versions of a hypothesis exist.
`Alternative SupportSOU`	`ALTERNATIVE_ SUPPORT_SOU`	200	0.5	Alternative evidence could play the same role as a current piece of support evidence.
`Uncertain SupportSOU`	`UNCERTAIN_ SUPPORT_SOU`	300	0.5	Support inference may be incorrect.
`Possible Alternative ExplanationSOU`	`POSSIBLE_ ALTERNATIVE_ EXPLANATION_SOU`	300	0.5	Explanations may exist that could play the same role as a current explanation.
`Alternative ExplanationSOU`	`ALTERNATIVE_ EXPLANATION_SOU`	300	0.5	More than one possible explanation exists.

Table C.5: Pre-defined concrete classes derived from `SOU` base class.

Base Class Name	*REQUIRED SLOTS*		
	name	*type*	*description*
NP_PLAN	ID	int	Class ID.
	GOAL_FORM	int	Subgoal met by this plan.
	INPUT_TYPE	string	Name of data type accepted as input to plan (must be a pointer type).
	OUTPUT_TYPE	string	Name of data type returned as output from plan (must be a pointer type).

Base Class Name	*REQUIRED METHODS*	
	name	*description*
NP_PLAN	SCHEMA	Plan schema.

Base Class Name	*OPTIONAL METHODS*		
	name	*description*	*default behavior*
NP_PLAN base class	IN_CONSTRAINTS	Predicate constraining values of input variables that plan can accept.	No constraints.
	PRECONDITION	Predicate which rules out plan applicability.	No condition.
	IN_BINDINGS	Set variables' values BEFORE schema begins.	Do nothing.
	VARIABLE_FOCUS	Select bindings for plan variables when several are available.	Bind to first input variable.
	SUBGOAL_FOCUS	Reorder subgoals when several may be satisfied in parallel.	Use order of posting.
	MATCH_FOCUS	Select among competing plans for satisfying this plan's subgoals.	Use the first plan found that matches.
	FAILURE	Predicate that returns TRUE when plan definitely cannot meet subgoal.	No failure condition.

Table C.6: IPUScript slot and method specifications for the NP_PLAN base class.

to refer to the plan class. Contiguous integers, starting with PLAN_ID+1, must be assigned to plan classes (both non-primitive and primitive) as shown in the example code given in Sec. C.3. The use of non-contiguous integer IDs will result in a run-time error. The GOAL_FORM slot declares the ID of the subgoal met by the plan. Only a single subgoal may be met by a plan class. The INPUT_TYPE and OUTPUT_TYPE slots contain strings describing the data type used as input and output by the plan. These must be pointer data types and must also be the same as those of the subgoal class whose ID is listed in GOAL_FORM. Violation of either of these conditions will result in a run-time error.

The plan schema is defined in the required SCHEMA method, which is described in detail in the following section. In the remainder of this section we shall discuss the optional methods and the inherited slots and methods of a derived non-primitive plan class.

The IN_CONSTRAINTS method allows restrictions to be placed on the input variable values for which the plan may be considered applicable. It is invoked by the planner as part of the *matching* process. When a subgoal is posted to the control planner, the planner searches for a plan which may potentially satisfy that subgoal. The first stage of matching looks for a plan that matches the subgoal in its GOAL_FORM slot. The second stage passes the input to the subgoal as input to the matching plans and checks the plan's IN_CONSTRAINTS. This method must return a boolean value indicating whether *any* of the inputs to the plan are acceptable. This allows the plan to rule itself out when no appropriate input values are available.

The PRECONDITION method performs a similar function to the IN_CONSTRAINTS. It is used to rule out the plan's applicability, but is based on criteria other than the contents of the input variables. It should be used to constrain the use of the plan based on the contents of the blackboard, the internal state of the planner, or other easily tested conditions. It returns a boolean value indicating whether the preconditions for the plan are met.

The IN_BINDINGS method performs initialization of the data members of the class. It is invoked after the input has been bound to a single value, just before the plan schema executes its first step. It provides the functionality normally associated with an class' constructor function.

The VARIABLE_FOCUS method converts a *plan instance* with a multi-valued input value (i.e. a list of potential inputs) to one or more *plan versions.* A plan version is a plan instance that has been bound to a single input value. For each input value for which a plan version is desired, the variable focus method may bind a new version using the BIND IPUScript macro. By default, a plan version is created only for the first element of the input variable list. It is currently recommended that only a single plan version be created for any given plan instance.

The SUBGOAL_FOCUS method is called by the planner when the plan schema posts multiple subgoals within a single step. In general, these subgoals may be considered to be satisfied in parallel. Subgoal focusing allows the subgoals to be re-ordered when some particular ordering is preferred. Subgoal focusing is performed by placing subgoal pointers from the local object named `subGoalList` of type `SubGoalPtrList` onto the object named `subGoalQueue` of type `SubGoalQueue`. Lists and queues are discussed in Sec. C.11.

Match focusing is the process by which a plan may influence which plans are selected to carry out the subgoals that it posts. After the plan matching function of the control planner checks the `IN_CONSTRAINTS` of all plans matching a subgoal, as described above, the `MATCH_FOCUS` method of the plan that posted the subgoal is passed the list of matching plans. It may then remove any matching plans that it deems unsuitable for meeting the subgoal. The default behavior is to select only the first matching plan available (i.e. the plan with the lowest ID). It is currently recommended that match focusing select no more than one plan to meet a single subgoal.

The `FAILURE` method provides a way for a non-primitive plan to halt itself if it determined that it definitely cannot meet its parent subgoal. It is called by the control planner after any primitive action is carried out. It must return a boolean value indicating whether it has failed. If it has, it is removed from the planning structure. The failure condition may be based upon internal plan data members, the blackboard state, or the internal state of the planner.

The public methods which follow may be invoked by other objects in the platform which obtain a pointer to the non-primitive plan object.

Public Methods

`InstanceID getInstanceID();`
: Return the unique instance identifier of the plan.

`uint goalForm();`
: Return the ID of the subgoal met by this plan.

`String inputType();`
: Return a string describing the data type accepted as input to the plan.

`uint isA();`
: Return the class ID.

`String nameOf();`
: Return the class name.

`void notify( SubGoalPtr subGoal );`
: Notify the plan of a significant event in the planner. If `subGoal` is `NULL`, the plan is halted and its status set to `ST_FAILED`. If `subGoal` is a pointer to a subgoal posted by the plan, the plan assumes the subgoal has completed and records its result and status. If `subGoal` is a pointer to the subgoal

that the plan is attempting to meet, the plan assumes that the subgoal has been met by other means or is no longer important, and the plan recursively halts all of its active subgoals (and their subsequent plans) and then halts with status `ST_FAILED`. If `subGoal` takes on any other value a run-time error is generated. This method is used extensively within the control planner, and may also be called by refocus units that wish to influence the flow of control in the planner.

`String outputType();`

Return a string describing the data type returned as output by the plan.

`int primitive();`

Always returns 1 for a non-primitive plan.

`uint planID();`

Return the class ID.

`StatusType status() const;`

Return the current status of the plan. Possible values are `ST_IDLE` (indicating that the plan schema has not yet been executed), `ST_RUNNING` (indicating that the plan schema has begun executing but has not yet completed), `ST_FINISHED` (indicating that the plan schema has finished satisfactorily), and `ST_FAILED` (indicating that the plan schema has been unable to meet its subgoal).

`uint subGoalCount();`

Return the number of subgoals posted by the plan that have not yet been satisfied.

C.7.2 Designing Plan Schemas

A plan schema is a method associated with a derived non-primitive plan class and has a very specialized structure. It encodes an algorithm for satisfying a subgoal as a series of *steps* which are performed sequentially. On each call to the `SCHEMA` method, a single step of the algorithm is executed. A step typically posts one or more subgoals to the planner. All non-primitive plan methods and IPUScript utility functions may be called within a schema. Subsequent schema steps are not executed until all subgoals posted by the plan have either been satisfied or have failed. The control planner will continue to execute schema steps until the completion status of the plan is set to `ST_FINISHED` or `ST_FAILED` using the `setStatus` method. The final schema step usually sets the status, as well as setting the plans output if necessary using the `setOutput` method. Failure to set the output value or final status of a plan schema will result in a run-time error.

Plan schema steps must be assigned contiguous integer step numbers starting from 1, and must appear in the schema definition starting with the IPUScript macro STEP(n). Here, n is the step number. Each schema step contains all statements between the STEP macro and the next STEP macro or END_DEFINE_NP_PLAN_ACTION. Improper numbering of schema steps will result in a run-time error. Examples of two non-primitive plans are shown below. They are followed by descriptions of the data members and methods inherited from the NP_PLAN class that may be used within derived methods.

```
// Code fragment from header file cat.h
    :
#define SGID_HAVE_PSM_INITIALIZED (SUBGOAL_TAG + 1)
#define SGID_HAVE_PSM_UNCERTAINTY_RESOLVED (SUBGOAL_TAG + 2)
#define SGID_HAVE_PSM_SOU_SELECTED (SUBGOAL_TAG + 3)
#define SGID_HAVE_PSM_SOU_RESOLVED (SUBGOAL_TAG + 4)
#define SGID_HAVE_SOLUTION (SUBGOAL_TAG + 5)
    :
#define PID_SOLVE_PROBLEM (PLAN_TAG + 1)
#define PID_INITIALIZE_PSM (PLAN_TAG + 2)
#define PID_RESOLVE_PSM_UNCERTAINTY (PLAN_TAG + 3)
#define PID_GET_SOLUTION (PLAN_TAG + 4)
    :
BEGIN_DECLARE_SEGMENT
    :
BEGIN_DECLARE_NP_PLAN( SolveProblem )
    SubGoalRef child;
END_DECLARE_NP_PLAN

BEGIN_DECLARE_NP_PLAN( ResolvePSMUncertainty )
    SubGoalRef child;
END_DECLARE_NP_PLAN
    :
// End of code fragment from file cat.h

// Code fragment from definition segment in source file cat.C
    :
DEFINE_NP_PLAN( SolveProblem, PID_SOLVE_PROBLEM, NONE,
                "IPUSString *", "PatchHyp *" )
BEGIN_DEFINE_NP_PLAN_SCHEMA( SolveProblem )
STEP(1)
```

```
    posti( SGID_HAVE_PSM_INITIALIZED, getInput() );
STEP(2)
    post( SGID_HAVE_PSM_UNCERTAINTY_RESOLVED );
STEP(3)
    child = post( SGID_HAVE_SOLUTION );
STEP(4)
    setOutput( getResult( child ) );
    setStatus( ST_FINISHED );
END_DEFINE_NP_PLAN_SCHEMA

DEFINE_NP_PLAN( ResolvePSMUncertainty, PID_RESOLVE_PSM_UNCERTAINTY,
                SGID_HAVE_PSM_UNCERTAINTY_RESOLVED, "", "" )
BEGIN_DEFINE_NP_PLAN_SCHEMA( ResolvePSMUncertainty )
STEP(1)
    child = post( SGID_HAVE_PSM_SOU_SELECTED );
STEP(2)
    PSMSOUPtr sou = (PSMSOUPtr) getResult( child );
    if ((sou->isA() == UNCERTAIN_ANSWER_SOU) && (sou->rating > 0.9)) {
        setStatus( ST_FINISHED );
    }
    else {
        posti( SGID_HAVE_PSM_SOU_RESOLVED, sou );
        NEXT_STEP(1);
    }
END_DEFINE_NP_PLAN_SCHEMA
    ⋮
// End of code fragment from file cat.C
```

Protected Data Members

`IPUS& environment;`

A reference to the IPUS derived class instance in which the plan is executing.

`SubGoal& parent;`

A reference to the SUBGOAL derived class instance which the plan is attempting to satisfy.

`VoidPtrList inputs;`

The multi-valued values of the inputs to the plan. These must be cast to the type specified as the INPUT_TYPE to the plan.

Protected Methods

`void addHypothesis( const LevelID level, HypothesisPtr hypothesis );`

Place `hypothesis` on the specified level of the blackboard.

`void *getInput();`
: Return the input variable to a plan version. The input must have already been bound using the `setInput` method, otherwise run-time error is generated.

`void *getResult( SubGoalRef subGoal );`
: Return the result from a previously posted subgoal.

`StatusType getStatus( SubGoalRef subGoal );`
: Return the status of a previously posted subgoal.

`SubGoalRef post( SubGoalID subGoal );`
: Used within plan schema to post a subgoal that must be met in order for this plan to succeed. This form of posting should be used to post a subgoal that does not take any input variables. The return value from this function is a unique subgoal identifier that may be used in subsequent schema steps in calls to `getResult` and `getStatus`. Be certain to store this identifier in a class data member as opposed to a local schema variable so that it will persist across schema steps. A `SubGoalRef` is not a pointer to the subgoal object.

`SubGoalRef posti( SubGoalID subGoal, void *input );`
: Same as `post` except an input value is passed to the subgoal.

`SubGoalRef posti( SubGoalID subGoal, VoidPtrList *inputs );`
: Same as `post` except a multi-valued input value is passed to the subgoal.

`void setInput( void *input );`
: Used in variable focusing to bind the plan to a single input value. The value may be subsequently accessed within the plan schema using `getInput`.

`void setOutput( void *output );`
: Used in the plan schema to bind the output from the plan to a value.

`void setStatus( StatusType status );`
: Used in the plan schema to set the final status of the plan instance. Should be set to either `ST_FINISHED` if the plan completed successfully, or `ST_FAILED` if the plan failed. Once the plan status is set, no further schema steps will be executed.

C.8 Subgoals

Subgoal objects are instantiated within the platform to represent a specific goal within a plan schema. The posting of a subgoal to the planner (with the schema `post` or `posti` methods) creates such an instance, which is then acted upon by the planner. The subgoal may hold a single or multi-valued input value which is subsequently passed to the matching plans which then attempt to satisfy it. A subgoal has few methods because its behavior is minimal. It serves primarily as a symbolic liaison between a plan schema and lower levels of the planning structure.

C.8.1 Deriving Subgoals

The IPUScript slot and method specifications for the base `SUBGOAL` class are given in Table C.7. The `ID` slot specifies a unique identifier for the derived subgoal class.

Base Class Name	*REQUIRED SLOTS*		
	name	*type*	*description*
SUBGOAL	ID	int	Class ID.
	INPUT_TYPE	string	Name of data type accepted as input to subgoal (must be a pointer).
	OUTPUT_TYPE	string	Name of data type returned as output from subgoal (must be a pointer).

Base Class Name	*REQUIRED METHODS*	
	name	*description*
SUBGOAL	-	-

Base Class Name	*OPTIONAL METHODS*		
	name	*description*	*default behavior*
SUBGOAL	SATISFACTION	Test if actions performed to satisfy another subgoal have also satisfied this subgoal.	No satisfaction condition.
	OUT_CONSTRAINTS	Predicate constraining results deemed satisfactory for subgoal to be met.	No constraints.

Table C.7: IPUScript slot and method specifications for the `SUBGOAL` base class.

As with plans, subgoals must be assigned contiguous integer IDs, starting from `SUBGOAL_ID+1`. Improper assignment of IDs will result in a run-time error. The most important element of a derived subgoal class are the `INPUT_TYPE` and `OUTPUT_TYPE` slots. These slots indicate the pointer data types used as input and output of the subgoal, and *must* be identical to the input and output types of all plans that match the subgoal in their GOAL_FORM slot. Any mismatch between subgoal and matching plan slots in this regard will result in a run-time error. An example of the derivation of a simple concrete subgoal class is shown below.

```
// Code fragment from header file cat.h
      ⋮
#define SGID_HAVE_PSM_INITIALIZED (SUBGOAL_TAG + 1)
      ⋮
BEGIN_DECLARE_SEGMENT
      ⋮
DECLARE_SUBGOAL( HavePSMInitialized )
      ⋮
// End of code fragment from file cat.h

// Code fragment from definition segment in source file cat.C
      ⋮
DEFINE_SUBGOAL( HavePSMInitialized, SGID_HAVE_PSM_INITIALIZED,
                "IPUSString *", "" )
      ⋮
// End of code fragment from file cat.C
```

Optional methods may be specified for a derived subgoal class which enhance its standard behavior. The SATISFACTION method of each active subgoal is called by the control planner after every primitive action executes. The method returns a boolean value indicating whether the subgoal has been met by indirect means (i.e. through the action of a plan other than those instantiated to meet this subgoal). The criteria for determining the satisfaction of a subgoal may be based on information on the blackboard, on information internal to the planner, or on means external to the IPUS application. The OUT_CONSTRAINTS method is called by the control planner when the subgoal's matching plan completes and returns an output value to the subgoal. It returns a boolean value indicating whether the output variables truly satisfy the subgoal, allowing additional constraints to be placed on its successful completion. If the output constraints are not met, the plan is considered to have failed, and returns the status ST_FAILED.

C.9 Primitive Plans

Like the non-primitive plan, a primitive plan object is instantiated by the planner in order to meet a higher level subgoal. A primitive plan differs from a non-primitive plan in that it meets that subgoal directly by performing some *primitive action* that instead of posting additional subgoals, performs inferences that create or extend hypotheses on the blackboard. In this way, the primitive plan is analogous to the *knowledge source* of the classical blackboard paradigm. Like the non-primitive plan, the primitive plan provides a variety of optional methods which constrain its applicability and assist the planner in its use. A number of inherited methods are also available for use in designing the primitive action methods. In the following sections we describe and demonstrate the process of deriving primitive plans and their action methods using IPUScript.

C.9.1 Deriving Primitive Plans

IPUScript specifications for the derivation of concrete primitive plan classes are given in Table C.8. The ID, GOAL_FORM, INPUT_TYPE, and OUTPUT_TYPE slots, as well as the

Base Class Name	*REQUIRED SLOTS*		
	name	*type*	*description*
P_PLAN	ID	int	Class ID.
	GOAL_FORM	int	Subgoal met by this plan.
	INPUT_TYPE	string	Name of data type accepted as input to plan (must be a pointer type).
	OUTPUT_TYPE	string	Name of data type returned as output from plan (must be a pointer type).

Base Class Name	*REQUIRED METHODS*	
	name	*description*
P_PLAN	ACTION	Primitive action.

Base Class Name	*OPTIONAL METHODS*		
	name	*description*	*default behavior*
P_PLAN	IN_CONSTRAINTS	Predicate constraining input variables that plan can accept.	No constraints.
	PRECONDITION	Predicate that tests plan applicability.	No condition.
	IN_BINDINGS	Set variables' values before action begins.	Do nothing.
	VARIABLE_FOCUS	Select bindings for plan variables when several are available.	Bind to first input variable.

Table C.8: IPUScript slot and method specifications for the P_PLAN base class.

IN_CONSTRAINTS, PRECONDITION, IN_BINDINGS, and VARIABLE_FOCUS methods, all are used in the same way as with the non-primitive plan base class. Their purpose, as well the criteria for their specification, are described in Sec. C.7.1. The ACTION method is discussed in the following section.

A variety of public methods are provided by the P_PLAN base class and are described below. They may be called by any system element that obtains a pointer to the plan object.

<u>**Public Methods**</u>

`InstanceID getInstanceID();`
Return the unique instance identifier of the plan.

`uint goalForm();`
Return the ID of the subgoal met by this plan.

`String inputType();`
Return a string describing the data type accepted as input to the plan.

`uint isA();`
Return the class ID.

`String nameOf();`
Return the class name.

`void notify( SubGoalPtr subGoal );`
Notify the plan of a significant event in the planner. If subGoal is NULL, the plan is halted and its status set to ST_FAILED. If subGoal is a pointer to the subgoal that the plan is attempting to meet, the plan assumes that the subgoal has been met by other means or is no longer important, and the plan halts with status ST_FAILED. If subGoal takes on any other value a run-time error is generated. This method is used extensively within the control planner, and may also be called by refocus units that wish to influence the flow of control in the planner.

`String outputType();`
Return a string describing the data type returned as output by the plan.

`int primitive();`
Always returns 0 for a primitive plan.

`uint planID();`
Return the class ID.

`StatusType status() const;`

Return the current status of the plan. Possible values are `ST_IDLE` (indicating that the plan schema has not yet been executed), `ST_RUNNING` (indicating that the plan schema has begun executing but has not yet completed), `ST_FINISHED` (indicating that the plan schema has finished satisfactorily), and `ST_FAILED` (indicating that the plan schema has been unable to meet its subgoal).

C.9.2 Designing Primitive Actions

The `ACTION` method of a concrete primitive plan class performs atomic actions representing the application of domain specific knowledge. This includes creating and extending blackboard hypotheses, making inferences based on those hypotheses, and producing intermediate problem solving steps related to active planning. They must contain the complete resources to meet their parent subgoal, because they cannot post additional subgoals to the planner.

The `ACTION` method takes the form of a "standard" C++ algorithm, and is executed from start to finish in one call (unlike the plan schema). This algorithm may include calls to external software modules, file and device I/O, the IPUScript utility functions, public methods of other object in the IPUS application, and the inherited methods listed below. The action performed by a primitive plan is, of course, directly dependent upon the subgoal that it was designed to meet.

By far the most common (and important) task performed within a primitive action is that of *inferencing*. An inference is the act of creating or changing a hypothesis based on evidence occurring in another hypothesis. A number of features are provided through the blackboard and hypothesis methods for tracking inferential relations, but *it remains the responsibility of the primitive action designer to ensure that the integrity of these features is maintained.* To this end, all inferencing actions must perform the following tasks.

Maintenance of Hierarchical Relations A hierarchy of relationships between hypothesis is maintained through the `supportList` and `explanationList` data members of hypotheses. As these relationships are created and evolve through the actions of primitive plans, it is the sole responsibility of those plans to ensure that the relationships are explicitly represented. This is accomplished by adding and removing elements from those lists.

Representation of Hypothesis Uncertainty As hypotheses are created and extended, their associated uncertainty changes in both kind and amount. It is the sole responsibility of the primitive actions that modify those hypotheses to maintain their uncertainty representations. This is performed by adding and removing SOUs from their `souList` data members.

<u>**Protected Data Members**</u>

`IPUS& environment;`

A reference to the `IPUS` derived class instance in which the plan is executing.

`SubGoal& parent;`

A reference to the `SUBGOAL` derived class instance which the plan is attempting to satisfy.

`VoidPtrList inputs;`

The multi-valued values of the inputs to the plan. These must be cast to the type specified as the `INPUT_TYPE` to the plan.

Protected Methods

`void addHypothesis( const LevelID level, HypothesisPtr hypothesis );`

Place `hypothesis` on the specified level of the blackboard.

`void *getInput();`

Return the input variable to a plan version. The input must have already been bound using the `setInput` method, otherwise run-time error is generated.

`void setInput( void *input );`

Used in variable focusing to bind the plan to a single input value. The value may be subsequently accessed within the plan schema using `getInput`.

`void setOutput( void *output );`

Used in the plan schema to bind the output from the plan to a value.

`void setStatus( StatusType status );`

Used in the plan schema to set the final status of the plan instance. Should be set to either `ST_FINISHED` if the plan completed successfully, or `ST_FAILED` if the plan failed. Once the plan status is set, no further schema steps will be executed.

C.10 Refocus Units

Refocus units provide the control planner with a *reactive* component. They allow the system to break away from the strategies directly encoded in the plans, and *refocus* the course of problem solving in situations where particular branches of the planning tree may not be making adequate progress towards a solution. Refocus units may be posted to the planner from any method belonging to a non-primitive plan, primitive plan, subgoal, or refocus unit using the postRefocusUnit utility function described in Sec. C.12. Once posted, they act as independent "daemons" which may be activated at some future time to alter the activities of the control planner. Typically, a refocus unit will prune some branch of the planning structure using the notify method of the plan at the top of the branch.

Table C.9 lists the IPUScript slot and method specifications for the REFOCUS_UNIT base class. Derived refocus units must be assigned contiguous in-

Base Class Name	*REQUIRED SLOTS*		
	name	*type*	*description*
REFOCUS_UNIT	ID	int	Class ID.

Base Class Name	*REQUIRED METHODS*	
	name	*description*
REFOCUS_UNIT	CONDITION	Predicate that indicates when to invoke handler.
	HANDLER	Performing refocusing operation.
	REMOVAL	Predicate that indicates when to remove refocus unit.

Base Class Name	*OPTIONAL METHODS*		
	name	*description*	*default behavior*
REFOCUS_UNIT	-	-	-

Table C.9: IPUScript slot and method specifications for the REFOCUS_UNIT base class.

teger IDs starting with REFOCUS_UNIT_ID+1. Incorrect numbering of IDs will result in a run-time error. Refocus units require the specification of three required methods which work in concert to give the class their daemon-like behavior. Once a refocus unit is instantiated and posted to the control planner, its CONDITION method is called after any primitive action completes. A boolean value must be returned indicating whether the refocus unit's action should be invoked. If the condition is met, the HANDLER method is immediately called to perform the refocus units action. Regardless of the condition value, the REMOVAL method is then invoked. It returns a boolean value indicating whether the refocus unit should be removed from the planner and deleted. An example of the derivation of a refocus unit is shown below.

```
// Code fragment from header file cat.h
      ⋮
#define RUID_TIMER_UNIT (REFOCUS_UNIT_ID + 1)
      ⋮
```

```
BEGIN_DECLARATION_SEGMENT
    ⋮
BEGIN_DECLARE_REFOCUS_UNIT( TimerUnit )
    PlanPtr poster;
    int timer;
END_DECLARE_REFOCUS_UNIT
    ⋮
// End of code fragment from file cat.h

// Code fragment from definition segment in source file cat.C
    ⋮
DEFINE_REFOCUS_UNIT( TimerUnit, RUID_TIMER_UNIT )
BEGIN_DEFINE_REFOCUS_UNIT_CONDITION( TimerUnit )
    return (--timer > 0);
END_DEFINE_REFOCUS_UNIT_CONDITION

BEGIN_DEFINE_REFOCUS_UNIT_HANDLER( TimerUnit, )
    poster->notify( NULL );
END_DEFINE_REFOCUS_UNIT_HANDLER

BEGIN_DEFINE_REFOCUS_UNIT_REMOVAL( TimerUnit, )
    return (timer <= 0);
END_DEFINE_REFOCUS_UNIT_REMOVAL
    ⋮
// End of code fragment from file cat.C
```

C.11 Lists and Iterators

List objects are used throughout the IPUS platform as a means for grouping like objects together. A list may contain only objects derived from a specific base class. The class restrictions associated with a list are specified within the class name of the list; i.e. a HypothesisPtrList can contain only pointers to Hypothesis instances and objects derived from Hypothesis. These restrictions are enforced by the C++ compiler. Lists are implemented in the IPUS platform using singly linked lists. They provide a built in iterator (see Sec. C.11.2) and support all iterator methods directly.

Lists come in two basic "flavors"—sorted and unsorted. A sorted list maintains the order of its elements according to a sorting criteria. An unsorted list allows elements of the list to assume any order. Sorted and unsorted lists have the same public methods, but the methods give slightly differing behaviors.

Since lists are type restrictive, their method prototypes define specific type parameters through the use of template classes. We show the prototypes here using the generic identifier `Object` to represent the object stored in the list.

Public Methods

`void add( Object& object );`
: *Unsorted lists:* Same as `addAfter`. *Sorted lists:* Add `object` to the list in its sorted position.

`void addAfter( Object& object );`
: *Unsorted lists:* Add `object` after the list element pointed to by the built-in iterator. *Sorted lists:* Add `object` to the list in its sorted position.

`void addFirst( Object& object );`
: *Unsorted lists:* Add `object` to the front of the list. *Sorted lists:* Same as `add`.

`void addLast( Object& object );`
: *Unsorted lists:* Add `object` to the end of the list. *Sorted lists:* Same as `add`.

`int find( Object& object );`
: Move the built-in iterator to the next list element that equates to `object`. Returns a boolean value indicating the success of the search. If the search is unsuccessful, the iterator is unmoved.

`void flush();`
: Remove all elements from the list.

`ObjectListIterator getIterator();`
: Return an iterator over the list that points to the first element.

`void goto( uint n );`
: Move the built-in iterator to the `n`th element of the list.

`void intersect( ObjectList anotherList );`
: Remove all elements from the list that are not also on `anotherList`.

`int isEmpty();`
: Return a boolean value indicating whether the list has any elements.

`int isEnd();`
: Return a boolean value indicating whether the built-in iterator is positioned "off the end" of the list.

`Object& item();`
: Return the list element pointed to by the built-in iterator. A run-time error is produced if the iterator is "off the end" of the list (see Sec. C.11.2).

`uint length();`
: Return the number of elements in the list.

`void next();`
: Advance built-in iterator to next list element.

`void remove();`
: Remove the list element pointed to by the built-in iterator from the list. If the list element is a pointer, the object referred to by the pointer is *not* deleted.

`void resetFirst();`
: Position the built-in iterator at the first element of the list.

`void resetLast();`
: Move the built-in iterator to the last element of the list.

`void set( Object& object );`
: *Unsorted lists:* Set the list element pointed to by the built-in iterator to `object` by assignment. *Sorted lists:* Remove the list elements pointed to by the built-in iterator and add `object` to the list.

C.11.1 Queues

Queues are a very simple form of an ordered but unsorted list. They have only four methods defined, supporting the classical model of FIFO queues. Iterators may not be used with queues.

Public Methods

`void enq( Object object );`
: Add `object` to the end of the queue.

`Object deq();`
: Remove the object at the head of the queue and return it to the caller.

`int isEmpty();`
: Return a boolean value indicating whether the queue is empty.

`void flush();`
: Remove all elements from the queue. If the queue is of pointer types, the objects referenced by the pointers are *not* deleted.

C.11.2 Iterators

Iterator objects are used throughout the IPUS platform for accessing lists of objects. They provide a means for viewing, modifying, and augmenting, the contents of lists that remains independent of the internal structure of the list. Each specific list class has its own specific iterator class—for example, a HypothesisList is accessed with a HypothesisListIterator. An iterator is an intelligent pointer into a list. At any time, it either points to an element of a list, or is "off the end" of the list. An iterator provides methods to move itself about the list, seek a specific element of the list and insert, change, and remove elements of the list. This section contains a list of iterator methods, and a description of their behaviors.

Since iterators refer explicitly to lists with elements of a particular class type, we show here the identifier `Object` as a generic type. Replace this with the name of the class of elements placed on the list to obtain the proper identifier or prototype.

Public Methods

`void add( Object& object );`
Unsorted lists: same as `addAfter`. *Sorted lists:* Add `object` to the list in its proper location.

`void addAfter( Object& object );`
Unsorted lists: Add `object` after the list element pointed to by the iterator. *Sorted lists:* Same as `add`.

`int find( Object& object );`
Move iterator to the next list element that equates to `object`. Returns a boolean value indicating the success of the search. If the search is unsuccessful, the iterator is unmoved.

`void goto( uint n );`
Move iterator to the `n`th element of the list.

`int isEnd();`
Return a boolean value indicating whether the iterator is positioned off the end of the list (i.e. not pointing to a list element).

`int isFirst();`
Return a boolean value indicating whether the iterator is pointing to the first element of the list.

`int isLast();`
Return a boolean value indicating whether the iterator is pointing to the last element of the list.

`Object& item();`
: Return the list element pointed to by the iterator.

`void next();`
: Move iterator to next element of list.

`void resetFirst();`
: Move iterator to first list element.

`void resetLast();`
: Move iterator to last list element.

`void set( Object& object );`
: Set the list item pointed to by the iterator to `object` by assignment.

`void remove();`
: Remove the list element pointed to by the iterator. If the list element is a pointer, the object referred to by the pointer is *not* deleted.

C.12 IPUScript Utility Functions

The IPUScript utility functions may be used within all methods belonging to concrete classes derived from the `NP_PLAN`, `P_PLAN`, `SUBGOAL`, and `REFOCUS_UNIT` base classes. They allow access to global system features, most of which are provided by the IPUS application object.

Utility Functions

`HypothesisPtrListIterator getAnswerListIterator();`
: Return an iterator over the answer list of the PSM.

`LevelIterator getLevelIterator( const LevelID level );`
: Return an iterator over the specified level of the blackboard.

`PSMSOUPtr getPSMSOU();`
: Return a pointer to the highest priority PSM SOU on the PSM SOU list.

`PSMSOUPtrListIterator getPSMSOUListIterator();`
: Return an iterator over the PSM SOU list.

`void installPSM( PSMPtr psm );`
: Install the PSM instance pointed to by `psm` into the IPUS application. Only one PSM may be installed at any time.

```
RefocusUnitPtr newRefocusUnit( const uint refocusUnitID );
```
Create a new instance of the refocus unit class specified by `refocusUnitID`. The refocus unit will not be used by the planner until it is posted to the planner using `postRefocusUnit`.

```
void postRefocusUnit( RefocusUnitPtr refocusUnit );
```
Post a refocus unit instance to the planner. The refocus unit will operate as described in Sec. C.10.

Bibliography

[1] V. Lesser, H. Nawab, M. Bhandaru, N. Carver, Z. Cvetanovic, I. Gallastegi and F. Klassner, "Integrated Signal Processing and Signal Understanding," Technical Report 91-34, Computer and Information Science Department, University of Massachusetts, 1991.

[2] S. H. Nawab and V. Lesser, "Integrated Processing and Understanding of Signals," in *Symbolic and Knowledge-Based Signal Processing*, eds. A. V. Oppenheim and S. H. Nawab, pp. 251-285, Prentice-Hall: Englewood Cliffs, N.J., 1992.

[3] N. Carver and V. Lesser, "A Planner for the Control of Problem-Solving Systems," *IEEE Trans. on Systems, Man, and Cybernetics,* v. 23, n. 6, pp. 1519-1536, 1993.

[4] B. W. Kernighan and D. M. Ritchie, *The C Programming Language, 2d Ed.,* Prentice Hall: Englewood Cliffs, N.J., 1988.

[5] M. A. Ellis and B. Stroustrup, *The Annotated C++ Reference Manual,* Addison-Wesley: Reading, MA, 1990.

[6] S. B. Lippman, *C++ Primer, 2d Ed.,* Addison-Wesley: Reading, MA, 1991.

Rome Laboratory

Customer Satisfaction Survey

RL-TR-________

Please complete this survey, and mail to RL/IMPS, 26 Electronic Pky, Griffiss AFB NY 13441-4514. Your assessment and feedback regarding this technical report will allow Rome Laboratory to have a vehicle to continuously improve our methods of research, publication, and customer satisfaction. Your assistance is greatly appreciated.
Thank You

Organization Name:________________________________(Optional)

Organization POC: ________________________________(Optional)

Address:____

1. On a scale of 1 to 5 how would you rate the technology developed under this research?

5-Extremely Useful   1-Not Useful/Wasteful

Rating_____

Please use the space below to comment on your rating. Please suggest improvements. Use the back of this sheet if necessary.

2. Do any specific areas of the report stand out as exceptional?

Yes___ No______

If yes, please identify the area(s), and comment on what aspects make them "stand out."

3. Do any specific areas of the report stand out as inferior?

Yes___ No___

If yes, please identify the area(s), and comment on what aspects make them "stand out."

4. Please utilize the space below to comment on any other aspects of the report. Comments on both technical content and reporting format are desired.

*U.S. GOVERNMENT PRINTING OFFICE: 1995-610-126-20050